UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

Cartographie, localisation et planification simultanées 'en ligne', à long terme et à grande échelle pour robot mobile

Thèse de doctorat
Spécialité : génie électrique

Mathieu LABBÉ

Sherbrooke (Québec) Canada

Septembre 2018

MEMBRES DU JURY

François MICHAUD
Directeur

Philippe GIGUÈRE
Évaluateur

Frédéric MAILHOT
Évaluateur

François POMERLEAU
Évaluateur

RÉSUMÉ

Pour être en mesure de naviguer dans des endroits inconnus et non structurés, un robot doit pouvoir cartographier l'environnement afin de s'y localiser. Ce problème est connu sous le nom de cartographie et localisation simultanées (ou SLAM pour *Simultaneous Localization and Mapping*). Une fois la carte de l'environnement créée, des tâches requérant un déplacement d'un endroit connu à un autre peuvent ainsi être planifiées. La charge de calcul du SLAM est dépendante de la grandeur de la carte. Un robot a une puissance de calcul embarquée limitée pour arriver à traiter l'information 'en ligne', c'est-à-dire à bord du robot avec un temps de traitement des données moins long que le temps d'acquisition des données ou le temps maximal permis de mise à jour de la carte. La navigation du robot tout en faisant le SLAM est donc limitée par la taille de l'environnement à cartographier.

Pour résoudre cette problématique, l'objectif est de développer un algorithme de SPLAM (*Simultaneous Planning Localization and Mapping*) permettant la navigation peu importe la taille de l'environment. Pour gérer efficacement la charge de calcul de cet algorithme, la mémoire du robot est divisée en une mémoire de travail et une mémoire à long terme. Lorsque la contrainte de traitement 'en ligne' est atteinte, les endroits vus les moins souvent et qui ne sont pas utiles pour la navigation sont transférées de la mémoire de travail à la mémoire à long terme. Les endroits transférés dans la mémoire à long terme ne sont plus utilisés pour la navigation. Cependant, ces endroits transférés peuvent être récupérées de la mémoire à long terme à la mémoire de travail lorsque le le robot s'approche d'un endroit voisin encore dans la mémoire de travail. Le robot peut ainsi se rappeler incrémentalement d'une partie de l'environment a priori oubliée afin de pouvoir s'y localiser pour le suivi de trajectoire.

L'algorithme, nommé RTAB-Map, a été testé sur le robot AZIMUT-3 dans une première expérience de cartographie sur cinq sessions indépendantes, afin d'évaluer la capacité du système à fusionner plusieurs cartes 'en ligne'. La seconde expérience, avec le même robot utilisé lors de onze sessions totalisant 8 heures de déplacement, a permis d'évaluer la capacité du robot de naviguer de façon autonome tout en faisant du SLAM et planifier des trajectoires continuellement sur une longue période en respectant la contrainte de traitement 'en ligne' . Enfin, RTAB-Map est comparé à d'autres systèmes de SLAM sur quatre ensembles de données populaires pour des applications de voiture autonome (KITTI), balayage à la main avec une caméra RGB-D (TUM RGB-D), de drone (EuRoC) et de navigation intérieur avec un robot PR2 (MIT Stata Center).

Les résultats montrent que RTAB-Map peut être utilisé sur de longue période de temps en navigation autonome tout en respectant la contrainte de traitement 'en ligne' et avec une qualité de carte comparable aux approches de l'état de l'art en SLAM visuel et avec télémètre laser. ll en résulte d'un logiciel libre déployé dans une multitude d'applications allant des robots mobiles intérieurs peu coûteux aux voitures autonomes, en passant par les drones et la modélisation 3D de l'intérieur d'une maison.

Mots-clés : Robotique, Cartographie et localisation simultanées, Gestion de mémoire

À tous mes supporteurs.

REMERCIEMENTS

Tout d'abord, mes recherches furent possibles grâce au support financier du Conseil de recherche en sciences et génie du Canada, la Fondation canadienne pour l'innovation, le programme des Chaires de recherche du Canada et le Fonds de recherche du Québec – Nature et technologies. Je voudrais également remercier la Faculté de génie pour l'octroie de la Médaille Léonard deVinci en 2016, ainsi qu'à l'équipe Maxed-Out, un groupe d'utilisateurs de l'environnement de programmation ROS de Silicon Valley, d'avoir choisi RTAB-Map pour gagner la compétition internationale de Kinect à la conférence *IEEE International Conference on Intelligent Robots and Systems* – IROS en 2014.

De plus, je veux remercier mon directeur de doctorat François Michaud pour sa grande confiance, son soutien, son enthousiaste ainsi que son intérêt pour mon projet. Finalement, je tiens à remercier toute l'équipe du laboratoire IntRoLab pour leur soutien technique et moral tout au long de mes travaux, en particulier les deux autres François, Dominic, Vincent, David, Francis, Michaël, Joël, Ronan, Aurélien, Guillaume et Antoine.

TABLE DES MATIÈRES

1 INTRODUCTION **1**

2 Loop Closure Detection for Multi-Session SLAM **5**
2.1 Introduction . . . 7
2.2 Online Multi-Session Graph-Based SLAM . . . 9
2.2.1 Loop Closure Detection . . . 9
2.2.2 Graph Optimization . . . 10
2.2.3 Memory Management for Online Multi-Session Mapping . . . 10
2.3 Results . . . 12
2.4 Discussion . . . 18
2.5 Conclusion . . . 21

3 Graph-Based SPLAM with Memory Management **23**
3.1 Introduction . . . 25
3.2 Related Work . . . 27
3.3 Memory Management for SPLAM . . . 28
3.3.1 Short-Term Memory Module . . . 32
3.3.2 Appearance-based Loop Closure Detection Module . . . 33
3.3.3 Proximity Detection Module . . . 34
3.3.4 Graph Optimization Module . . . 36
3.3.5 Path Planning Modules . . . 36
3.3.6 Patrol Module . . . 41
3.4 Results . . . 42
3.4.1 Influences of MM on SPLAM . . . 44
3.4.2 TPP-MPP Interactions . . . 46
3.4.3 Influences of LTM on TPP . . . 47
3.5 Discussion . . . 52
3.6 Conclusion . . . 55

4 RTAB-Map as an Open-Source SLAM Library **57**
4.1 Introduction . . . 59
4.2 Popular SLAM Approaches Available on ROS . . . 62
4.3 RTAB-Map Description . . . 67
4.3.1 Odometry Node . . . 69
4.3.2 Synchronization . . . 75
4.3.3 STM . . . 76
4.3.4 Loop Closure and Proximity Detection . . . 79
4.3.5 Graph Optimization . . . 80
4.3.6 Global Map Assembling . . . 81
4.4 Evaluating Trajectory Performance of RTAB-Map . . . 82
4.4.1 KITTI . . . 84

4.4.2 TUM 87
4.4.3 EuRoC 89
4.4.4 MIT Stata Center 92
4.5 Evaluating Computation Performance between Visual and Lidar SLAM Configurations with RTAB-Map 98
4.5.1 Examining the Use of RTAB-Map's Memory Management Mechanism 103
4.6 Discussion 105
4.7 Conclusion 109

5 CONCLUSION **111**

LISTE DES RÉFÉRENCES **113**

LISTE DES FIGURES

2.1 Memory management model. 11
2.2 Illustration of a local map created from multi-session mapping. 12
2.3 AZIMUT-3 robot equipped with a URG-04XL laser range finder and a Kinect sensor. 13
2.4 Resulting local maps without (left) and with (right) graph optimizations. . 14
2.5 Results for Map 4 and Map 5. 15
2.6 Processing time in relation to the number of nodes processed over time. . . 16
2.7 Top view of the map without optimization after five mapping sessions. . . 16
2.8 Loop closures between the mapping sessions. 17
2.9 Five online mapping sessions merged together automatically. 17
2.10 Graphs optimized. 19
2.11 Global maps 19
2.12 Processing time for each node added to graph. 20

3.1 The AZIMUT-3 robot equipped with a URG-04LX laser range finder and a Xtion PRO LIVE sensor. 29
3.2 Memory management and control architecture of SPLAM-MM. 29
3.3 Illustration of the local map and the global map in multi-session mapping. 32
3.4 Illustration of the role of the Proximity Detection module. 35
3.5 Illustration of how proximity detection works. 37
3.6 Example of obstacle detection using the laser rangefinder and the RGB-D camera. 39
3.7 Interaction between TPP and MPP for path planning. 41
3.8 Waypoints WP1 to WP4 identified on the global map. The purple path is the first path planned by TPP from the WP4 to WP1. 43
3.9 Global maps, optimized and not optimized, after reaching WP1. 44
3.10 Events that occurred during the trials. 45
3.11 Example of the effect of memory management. 45
3.12 Comparison of the corresponding images between the waypoint and at the last pose reached. 47
3.13 Memory size and total processing time over the 11 mapping sessions. . . . 48
3.14 Example of poses sent by TPP to MPP. 49
3.15 Example where MPP plans a slightly different path than the one provided by TPP. 50
3.16 Three examples illustrating how the graph reduction algorithm works. . . . 51
3.17 Comparison between the global maps. 53
3.18 Comparison of TPP planning time and LTM size. 54
3.19 Comparison of hard drive usage. 54

4.1 Block diagram of *rtabmap* ROS node. 68
4.2 Block diagram of *rgbd_ odometry* and *stereo_ odometry* ROS nodes. 70

4.3 Block diagram of *icp_odometry* ROS node. 73
4.4 Visual SLAM with a RGB-D camera like the Kinect for Xbox 360. 77
4.5 Visual SLAM with a stereo camera like the BumbleBee2. 77
4.6 Synchronization example of a RGB-D camera with laser scan and odometry. 78
4.7 STM's local occupancy grid creation. 79
4.8 Global map assembling. 82
4.9 Trajectories using RTAB-Map for three KITTI sequences. 85
4.10 Trajectories using RTAB-Map for three TUM sequences. 88
4.11 Trajectories using RTAB-Map for three EuRoC sequences. 91
4.12 Trajectories using RTAB-Map for Stata Center sequences. 94
4.13 Comparison of RTAB-Map with other lidar-based SLAM approaches 97
4.14 Local occupancy grid examples. 100
4.15 3D local occupancy grid map with ray tracing examples 101
4.16 2D occupancy grid map examples. 102
4.17 OctoMap using RGB-D camera. 103
4.18 Processing time required for each module inside *rtabmap*. 105
4.19 Global maps with and without memory management. 106

LISTE DES TABLEAUX

3.1 Parameters used for the trials 43

4.1 Popular ROS-compatible lidar and visual SLAM approaches 66
4.2 RTAB-Map (version 0.16.3) Default Parameters 83
4.3 ATE (m) results for the KITTI sequences 86
4.4 Average translational error (%) results for the KITTI sequences 87
4.5 Current state of KITTI's odometry leaderboard 88
4.6 ATE (cm) results for the TUM sequences 90
4.7 ATE (cm) results for the EuRoC sequences 90
4.8 Online results for the MIT Stata Center sequences 95
4.9 ATE (m) results of RTAB-Map on 2012-01-25 sequences. 99
4.10 Occupancy grid performance 99

CHAPITRE 1

INTRODUCTION

La cartographie et localisation simultanée (SLAM) [Stachniss *et coll.*, 2016] est une approche utilisée en robotique mobile lorsqu'un système de localisation externe comme le GPS n'est pas accessible pour estimer la position précise du robot dans l'environnement. Avec le SLAM, la position du robot est estimée à partir de ses capteurs en relation avec une carte de l'environment construite en même temps. Puisque les capteurs du robot ne sont pas parfaits, des erreurs dans la carte sont introduites, influençant du même coup la précision de la localisation. Pour corriger ces erreurs, le robot doit être en mesure de reconnaître lorsqu'il revient dans un endroit déjà visité : il détecte alors ce qui est qualifié être une fermeture de boucle. Une fois la fermeture de boucle détectée, un algorithme d'optimisation peut être utilisé pour propager l'erreur accumulée dans toute la carte afin de la corriger. La carte générée et la position connue dans celle-ci permettent ensuite de planifier des trajectoires sans collision lors de la navigation.

Pour pourvoir planifier de nouvelles trajectoires pendant que le robot navigue, l'algorithme de SLAM doit respecter la contrainte de traitement 'en ligne', c'est -à-dire que le temps de traitement des données doit être moins long que le temps d'acquisition des données ou du temps maximal permis de mise à jour de la carte. Par exemple, si le temps de traitement maximal est fixé à 1 seconde, l'algorithme doit être en mesure de toujours fournir une carte à jour à une fréquence d'au minimum 1 Hz. Lorsqu'un robot fait continuellement du SLAM, la carte construite sera de plus en plus grande si l'environnement n'est pas borné. Ou encore si l'environment est dynamique, il se peut que l'algorithme de SLAM duplique certains endroits [Glover *et coll.*, 2010] ou ait besoin de garder en mémoire plusieurs versions du même endroit pour maximiser la capacité de se localiser par la suite [Churchill et Newman, 2013]. Par exemple, si l'intensité de la lumière change au cours de la journée et que l'algorithme de localisation est basée sur les images, il se peut que garder plusieurs versions du même endroit en mémoire soit bénéfique pour être en mesure de se localiser à la fois le jour et la nuit.

La quantité de mémoire requise par le SLAM peut donc croître indéfiniment. Plus la carte est grande, plus de temps de traitement requis est long pour détecter les fermetures de boucles et pour corriger la carte. Il est donc important d'avoir un mécanisme de gestion de mémoire pour éviter d'avoir à traiter toutes les données à chaque mise à jour de la carte.

Une façon naïve de gestion de mémoire peut être d'éliminer les données les plus vieilles en premier. Le problème avec cette approche est que le robot ne pourra jamais par la suite se re-localiser ou planifier des trajectoires dans cette partie de l'environment. Une gestion de mémoire plus intelligente doit donc être faite pour oublier temporairement des endroits non essentielles à navigation courante afin de limiter le temps de mise à jour de la carte tout en pouvant se rappeler d'anciens endroits lorsque le robot retourne vers ceux-ci. La question de recherche résultante est la suivante : comment intégrer une gestion de mémoire intelligente à un algorithme de SLAM qui permet de respecter la contrainte de traitement 'en ligne' à long terme, tout en gardant assez d'information afin d'être en mesure de se localiser globalement et de naviguer dans tous les endroits connus ?

L'approche de gestion de la mémoire présentée dans cette thèse a pour but de réaliser un algorithme de SLAM satisfaisant la contrainte de traitement 'en ligne' pour un fonctionnement 'en ligne' à grande échelle et à long terme. Le temps de traitement, c.-à-d. le temps requis pour traiter une image acquise, est le critère utilisé pour limiter le nombre d'endroits conservés dans la mémoire de travail du robot. Pour identifier les endroits à conserver dans la mémoire de travail, la solution présentée consiste à conserver les endroits les plus récents et les plus fréquemment observés dans la mémoire de travail, et à transférer les autres dans la mémoire à long terme. Lorsqu'une correspondance est trouvée entre l'endroit actuel et un autre emmagasiné dans la mémoire de travail, les endroits voisins mémorisés dans la mémoire à long terme peuvent être retransférés en mémoire de travail afin de permettre la localisation dans des endroits précédemment "oubliés". Cette idée est inspirée d'observations faites en psychologie [Baddeley, 1997; Shiffrin, 2003] selon lesquelles les personnes se souviennent davantage des endroits où elles ont passé la plus grande partie de leur temps, comparativement à ceux dont elles ont vus moins souvent. En suivant cette heuristique, le compromis entre le temps et l'espace de recherche est donc fonction de l'environnement et des expériences du robot.

Une première version de cet algorithme de gestion de mémoire a été présentée dans [Labbé et Michaud, 2013] pour le problème spécifique de détection de fermeture de boucle à long terme et à grande échelle. L'approche de détection de fermeture de boucle est basée sur un filtrage bayésien pour estimer les hypothèses de fermeture de boucle. La vraisemblance entre le nouvel et les anciens endroits est calculée par l'approche de sac-de-mots (ou BOW pour *bag-of-words*). Des repères visuels sont extraits des images et ils sont ensuite quantifiés dans un dictionnaire incrémental de mots visuels. Chaque mot visuel dans le dictionnaire garde une liste des images dans lesquelles il se retrouve. Cet index inversé permet ensuite de comparer très rapidement une image avec une grande banque d'images selon un principe

de vote. Pour chaque mot visuel extrait dans l'image courante, les images dans la carte contenant le même mot vont recevoir un vote. L'image avec le plus de votes est la plus similaire à l'image actuelle. Le filtre bayésien estime aussi la probabilité que la nouvelle image provienne d'un nouvel endroit. Si la probabilité de fermeture de boucle d'un endroit connu est au-dessus un seuil prédéfini, une fermeture de boucle est alors détectée, sinon l'image est considérée provenir d'un nouvel endroit. La gestion de mémoire était utilisée pour limiter le temps de mise à jour du filtre bayésien et du dictionnaire de mots visuels afin d'être en mesure de détecter les fermetures de boucle toujours 'en ligne'. L'approche avait été testée seulement sur des ensembles de données d'images (réelles et synthétiques), il restait à étendre et expérimenter les principes sous-jacents à cet algorithme sur un vrai robot faisant du SLAM et de la navigation 'en ligne'. Dans cette thèse, ceux-ci sont intégrés à un système complet de SLAM et rigoureusement testé sur des robots. De plus, de nouveaux critères de gestion de mémoire ont été ajoutés pour que la navigation autonome dans des endroits a priori oubliés soit possible tout en respectant la contrainte de traitement 'en ligne'.

Le chapitre 2 montre, dans un premier temps, comment la gestion de mémoire peut être intégrée à un algorithme de SLAM sur un robot réel, et ce, dans un contexte multi-sessions, c'est-à-dire combinant plusieurs sessions de SLAM à partir de points de départ différents mais dont les endroits visités se recoupent afin de relier les cartes résultantes de ces sessions. Le chapitre 3 présente ensuite l'algorithme de cartographie, localisation et planification simultanées 'en ligne', à long terme et à grande échelle pour robot mobile, soit la contribution centrale de cette thèse. Le chapitre 4 termine en décrivant la librairie de SLAM résultante de la thèse, appelée RTAB-Map, distribuée comme logiciel libre et utilisée par des centaines de développeurs en robotique mobile. En plus de situer RTAB-Map par rapport à l'état de l'art en SLAM et d'évaluer les performances selon le coût et la sorte de capteurs utilisés, une comparaison exhaustive et juste entre les paradigmes de SLAM, visuel versus géométrique, pour la navigation autonome est faite sur un même robot, ce qui représente une première dans le domaine de la robotique mobile.

CHAPITRE 2

Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM

Avant-propos

Auteurs et affiliations :

M. Labbé : étudiant au doctorat, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique.

F. Michaud : professeur, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique.

Date d'acceptation : 17 mai 2014

État de l'acceptation : version finale publiée (`https://doi.org/10.1109/IROS.2014.6942926`)

Conférence : *IEEE/RSJ International Conference on Intelligent Robots and Systems*

Référence : [Labbé et Michaud, 2014]

Titre français : Détection de fermeture de boucle 'en ligne' pour cartographie et localisation simultanées à grande échelle et multi-session

Contribution au document : Cet article contribue à la thèse en élaborant comment une gestion de mémoire intelligente peut être intégrée à un système de SLAM complet pour permettre de cartographier, 'en ligne', un grand environment sur plusieurs sessions.

Résumé français : Pour de la cartographie et localisation simultanées à grande échelle et à long terme (SLAM), un robot doit faire face à un positionnement initial inconnu provoqué par le problème du robot kidnappé (c'est-à-dire le déplacement non-référencé du robot dans l'espace) ou parce que la carte est construite en plusieurs sessions. Cet article aborde ces problèmes en utilisant une approche de détection de fermeture de boucle globale, qui gère intrinsèquement ces situations, au SLAM. Cependant, la charge de calcul pour les approches de détection de fermeture de boucle globale est généralement influencée par la taille de l'environnement. L'approche de SLAM résultante, basé sur un graphe, utilise

alors une gestion de la mémoire qui considère uniquement certaines parties de la carte pour satisfaire aux exigences de traitement 'en ligne'. L'approche est évaluée sur cinq sessions de cartographie à l'intérieur d'un bâtiment avec un robot équipé d'un télémètre laser et d'une caméra Kinect.

Abstract

For large-scale and long-term simultaneous localization and mapping (SLAM), a robot has to deal with unknown initial positioning caused by either the kidnapped robot problem or multi-session mapping. This paper addresses these problems by tying the SLAM system with a global loop closure detection approach, which intrinsically handles these situations. However, online processing for global loop closure detection approaches is generally influenced by the size of the environment. The proposed graph-based SLAM system uses a memory management approach that only consider portions of the map to satisfy online processing requirements. The approach is tested and demonstrated using five indoor mapping sessions of a building using a robot equipped with a laser rangefinder and a Kinect.

2.1 Introduction

Autonomous robots operating in real life settings must be able to navigate in large, unstructured, dynamic and unknown spaces. To do so, they must build a map of their operating environment in order to localize itself in it, a problem known as Simultaneous localization and mapping (SLAM). A key feature in SLAM is detecting previously visited areas to reduce map errors, a process known as loop closure detection. Our interest lies with graph-based SLAM approaches [Lu et Milios, 1997] that use nodes as poses and links as odometry and loop closure transformations.

While single session graph-based SLAM has been largely addressed [Bosse *et coll.*, 2004; Grisetti *et coll.*, 2010; Thrun et Montemerlo, 2006], multi-session SLAM involves having to deal with the fact that robots, over a long period of operation, will eventually be shutdown and moved to another location without knowing it. Such situations include the so-called kidnapped robot problem and the initial state problem: when it is turned on, a robot does not know its relative position to a map previously created. One way to do multi-session mapping is to have the robot, on startup, localize itself in a previously-built map. This solution has the advantage to always use the same referential and only one map is created across the sessions. However, the robot must start in a portion of the environment already mapped, otherwise it never can relocalize itself in it. Another approach is to initialize a new map with its own referential and when a previously visited location is encountered, the transformation between the two maps can be computed. In [McDonald *et coll.*, 2012], special nodes called “anchor nodes" are used to keep transformation information between the maps. A similar approach is also used with multi-robot mapping [Kim *et coll.*, 2010]:

transformations between maps are computed when a robot sees the other or when a landmark is seen by both robots in their respective maps.

Global loop closure detection approaches, by being independent of the robot's estimated position [Ho et Newman, 2006], can intrinsically solve the problem of determining when a robot comes back to a previous map using a different referential [Cummins et Newman, 2011]. Popular global loop detection approaches are appearance-based [Angeli *et coll.*, 2008; Booij *et coll.*, 2009; Botterill *et coll.*, 2011; Konolige *et coll.*, 2010], exploiting the distinctiveness of images. The underlying idea behind these approaches is that loop closure detection is done by comparing all previous images with the new one. When loop closures are found between the maps, a global graph can be created by combining the graphs from each session. Graph pose optimization approaches [Folkesson et Christensen, 2007; Grisetti *et coll.*, 2007a; Johannsson *et coll.*, 2012] can then be used to reduce odometry errors using poses and link transformations inside each map and also between the maps.

All the solutions above can be integrated together to create a functional graph-based SLAM system. However, for loop closure detection and graph optimization approaches, online constraint satisfaction is limited by the size of the environment. For large-scale and long-term operation, the bigger the map is, the more computing power is required to process the data online. Mobile robots have limited computing resources, therefore online map updating is limited, and so some parts of the map must be somewhat forgotten. Memory management approaches [Labbé et Michaud, 2013] can be used to limit the size of the map so that loop closure detections are always processed under a fixed time limit, thus satisfying online requirements for long-term and large-scale environment mapping.

The solution presented in this paper simultaneously addresses these two problems: multi-session mapping, and online map updating with limited computing resources. Global loop closure detection is used across the mapping sessions to detect when the robot revisits a previous map. Using these loop closure constraints, the graph is optimized to minimize trajectory errors and to merge the maps together in the same referential. A memory management mechanism is used to limit the data processed by global loop closure detection and graph optimization in order to respect online constraints independently of the size of the environment. The algorithm is tested over five mapping sessions using a robot in an indoor environment.

The paper is organized as follows. Section 4.3 describes our approach. Section 2.3 presents experimental results and Section 4.6 discusses limitations of the approach on very long-term operation. Section 4.7 concludes the paper.

2.2 Online Multi-Session Graph-Based SLAM

In our approach, the underlying structure of the map is a graph with nodes and links. The nodes save odometry poses for each location in the map. The nodes also contain visualization information like laser scans, RGB images, depth images and visual words [Sivic et Zisserman, 2003] used for loop closure detection. The links store rigid geometrical transformations between nodes. There are two types of links: neighbor and loop closure. Neighbor links are added between the current and the previous nodes with their odometry transformation. Loop closure links are added when a loop closure detection is found between the current node and one from the same or previous maps. Our contribution in this paper involves combining two algorithms, loop closure detection [Labbé et Michaud, 2013] and graph optimization [Grisetti *et coll.*, 2007a], through a memory management process [Labbé et Michaud, 2013] that limits the number of nodes available from the graph for loop closure detection and graph optimization, so that they always satisfy online requirements.

2.2.1 Loop Closure Detection

For global loop closure detection, the bag-of-words approach described in [Labbé et Michaud, 2013] is used. Briefly, this approach uses a bayesian filter to evaluate loop closure hypotheses over all previous images. When a loop closure hypothesis reaches a pre-defined threshold H, a loop closure is detected. Visual words, which are SURF features quantized to an incremental visual dictionary, are used to compute the likelihood required by the filter.

In this paper, the RGB image, from which the visual words are extracted, is registered with a depth image, i.e., for each 2D point in the RGB image, a 3D position can be computed using the calibration matrix and the depth information given by the depth image. The 3D positions of the visual words are then known. When a loop closure is detected, the rigid transformation between the matching images is computed by a RANSAC approach using the 3D visual word correspondences. If a minimum of I inliers are found, loop closure is accepted and a link with this transformation between the current node and the loop closure hypothesis node is added to the graph. If the robot is constrained to operate on a single plane, the transformation can be refined with 2D iterative-closest-point (ICP) optimization [Besl et McKay, 1992] using laser scans contained in the matching nodes.

2.2.2 Graph Optimization

TORO [Grisetti *et coll.*, 2007a] (Tree-based netwORk Optimizer) is the graph optimization approach used, in which node poses and the link transformations are used as constraints. When loop closures are found, the errors introduced by the odometry can then be propagated to all links, thus correcting the map. It is relatively straightforward to use TORO to create a tree from the map's graph when there is only one map: the TORO tree has therefore only one root. In multi-session mapping, the different maps created have their own root with their own reference frames. When loop closures occur between the maps, TORO cannot optimize the graph if there are multiple roots. It may also be difficult to find a unique root if some portions of the map are forgotten or unavailable at that time (because of the memory management approach used to satisfy online processing requirements, explained in Sect. 2.2.3). To alienate these problems, our approach takes the root of the tree to be the latest node added to the current map graph, which is always uniquely defined across intra-session and inter-session mapping.

2.2.3 Memory Management for Online Multi-Session Mapping

For online mapping, new incoming data must be processed faster than the time required to acquire them. For example, if data are acquired at 1 Hz, new data should be added to the graph with global loop closure detection and graph optimization should be done in less than $R = 1$ second. The problem is that the time required for loop closure detection and graph optimization depends on the map's graph size. Long-term and large-scale online mapping is then limited by the size of the environment. To handle this, the RTAB-Map memory management approach [Labbé et Michaud, 2013] is used to maintain a graph manageable online by the loop closure detection and graph optimization algorithms, thus making the metric SLAM approach presented in this paper independent of the size of the environment.

The approach works as follows. The memory is composed of a Short-Term Memory (STM), a Working Memory (WM) and a Long-Term Memory (LTM), as shown by Figure2.1. The STM is the entry point for new nodes added to the graph when new data are acquired, and has a fixed size S. Nodes in STM are not considered for loop closure detection because they are generally very similar from one to another. When the STM size reaches S nodes, the oldest node is moved to WM to be considered for loop closure detection. The WM size indirectly depends on a fixed time limit T. When the time required to process the new data reaches T, some nodes of the graph are transferred from WM to LTM, thus keeping the WM size nearly constant. The LTM is not used for loop closure detection and

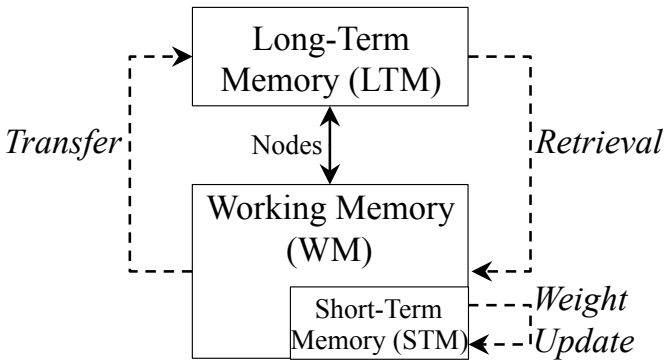


Figure 2.1 Memory management model.

graph optimization. However, if a loop closure is detected, neighbors in LTM of the old node can be transferred back to WM (a process called Retrieval) for further loop closure detections. In other words, when a robot revisits an area which was previously forgotten, it can remember incrementally the area if a least one node of this area is still in WM.

The choice of which nodes to keep in WM is based on a Weight Update step done in STM. The heuristic used to increase the weight of a node is based on the principle that, as humans do [Atkinson et Shiffrin, 1968; Baddeley, 1997], the robot should remember more the areas where they spent most of their time in. Therefore, the longer the robot is at a particular location, the larger the weight of the node should be. If two consecutive images are similar, i.e., the ratio of corresponding visual words between the images is over a specified threshold Y, the node's weight of the first image is increased by one and no new node is created for the second image. By following this heuristic, the compromise made between search time and space is therefore driven by the environment and the experiences of the robot. Oldest and less weighted nodes in WM are transferred to LTM before the others, thus keeping in WM only the nodes seen for longer periods of time.

For the approach presented in this paper, a local map consists of the biggest fully connected graph that can be created through neighbor and loop closure links from the last node (used as the root) with those in WM. Figure2.2 illustrates the concept. The diamonds represent initial and end nodes for each mapping session. The nodes in LTM are shown in red and the others are those in WM. The current local map is created and optimized only using nodes in WM that are linked to the last node (all nodes in the dashed area). The local map therefore represents more than the latest mapping session: it can span over multi-session mapping through loop closure links (green links). The other nodes still in WM that are not included in the local map are unreachable from the last node through links available in WM at this time.

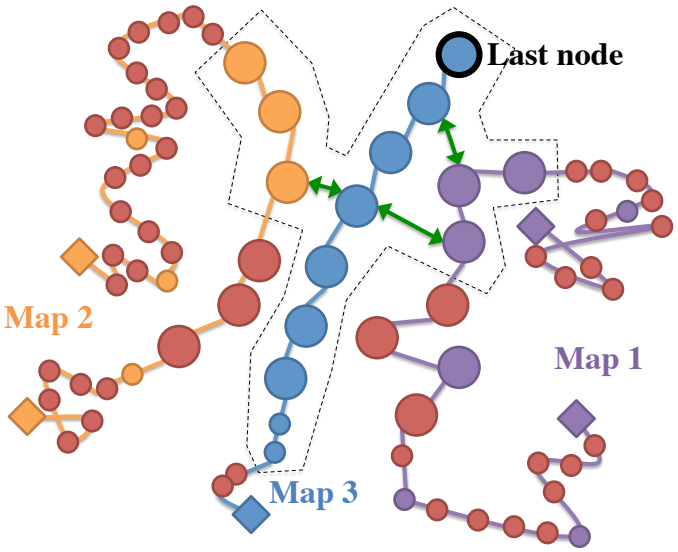


Figure 2.2 Illustration of a local map created from multi-session mapping.

Using this memory management approach, some parts of the map may be missing for graph optimization, as described in 2.2.2. Online graph optimization is done on the local map, with the constraints available in WM at that time. Constraints transferred to LTM are not used, thus limiting graph quality compared to using all constraints available. This is the compromise to make to be able to satisfy online processing requirements. However, if required, the approach is still able to create a global map by using all constraints from LTM and conduct offline a global graph optimization.

2.3 Results

The data sets used for the experiments are acquired using the AZIMUT-3 robot [Ferland *et coll.*, 2010], shown by Figure2.3, equipped with a URG-04LX laser rangefinder and a Kinect sensor. The RGB images from the Kinect are used for the appearance-based loop closure detection while the depth images are used to find the 3D position of the visual words. Laser scans and RGB-D point clouds created from the Kinect are used for map visualization. As mentioned in 2.2.1, since in this experiment the robot is constrained to a single plane, loop closure transformations are refined using 2D ICP with the laser scans to increase precision: the transformations are then limited to three degrees of freedom (x, y and rotation over z axis), ignoring noise on other degrees of freedom computed by the visual transformation.

Five mapping sessions (total length of 750 m) were conducted by starting the robot at different locations in our lab building. Between the mapping sessions, the robot was turned

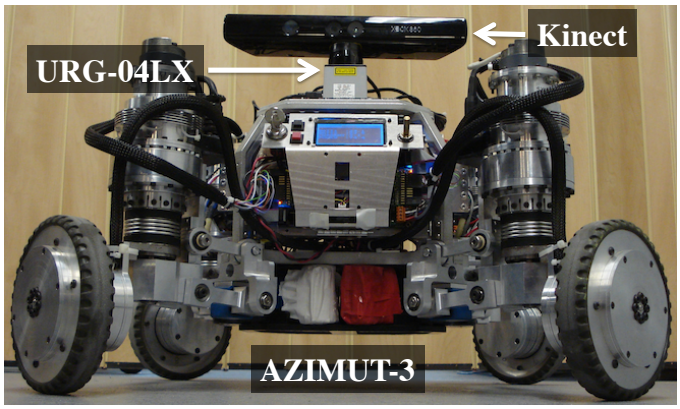


Figure 2.3 AZIMUT-3 robot equipped with a URG-04XL laser range finder and a Kinect sensor.

off to reset odometry, and moved to another location. In each session, the robot revisited at least one part of the environment mapped in a previous session. Data acquisition is done using the ROS bag mechanism (http://ros.org). Odometry, laser scans, RGB images and depth images are recorded at 1 Hz (i.e., $R = 1$ s) in a ROS bag. A ROS bag can be played using the same timings as during acquisition, making a realistic input for mapping and a good common format for other algorithms using ROS. One ROS bag per mapping session is taken. The ROS bags are processed on a MacBook Pro 2010: 2.66 GHz Intel Core i7 and SSD hard drive (on which the LTM is saved).

Two experiments were conducted (STM size $S = 10$, minimum inliers $I = 5$ of RANSAC, hypothesis threshold $H = 0.11$ and similarity threshold $Y = 0.45$). For the first experiment, our approach processed each mapping session independently, i.e., the memory was cleared between each session. Time limit T was set to 0.7 s. Figure2.4 shows the resulting maps for sessions 1, 2 and 3, with and without graph optimizations. The light gray areas are empty spaces detected using the laser rangefinder. No nodes were transferred to LTM in these experiments (local maps are equal to global maps). This is confirmed by Figure2.6: T was never reached for these sessions, and thus all nodes were used for loop closure detection and graph optimization. Figure2.5 shows results for the mapping sessions 4 and 5 (i.e., Map 4 and Map 5): the global graph not optimized (left), the last local map (middle) and the global map (right). The local map is the biggest map that was created online from the last node (with nodes available in WM), and the global map was generated offline after the mapping sessions (with all nodes in WM and LTM). As shown by Figure2.6, T was reached before the end. Figure2.5 b) illustrate the effect of transferring nodes to LTM to satisfy the online requirement. Even if loop closures can be detected with older portions of the map still in WM (as shown in a)), the maps cannot

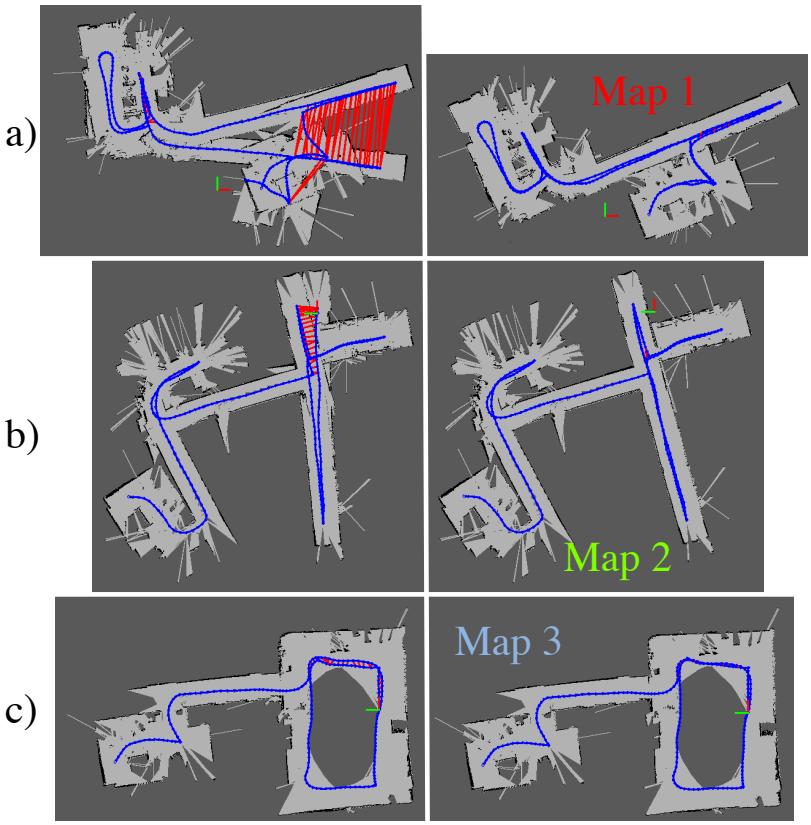


Figure 2.4 Resulting local maps without (left) and with (right) graph optimizations for a) Map 1, b) Map 2 and c) Map 3. Loop closures are shown in red.

be globally optimized if the neighbors of the loop closures are in LTM. For comparison, Figure2.5 c) are maps created offline using all constraints in LTM: here, loop closures with old portions of the map have an effect on graph optimization.

For the second experiment, the data sets for the five maps were processed one after each other, as in a real multi-session mapping trial. The robot automatically started a new map when the odometry was reset to zero before each session. The memory was preserved between the sessions and T was also set to 0.7 s. Figure2.7 shows the last local map (nodes in light gray areas are those in WM) and global graph (blue line) without optimization. The maps lie over each other because they are all starting from the same referential. Loop closures detected in the same map (intra-session) and those detected between the maps (inter-session) are shown in red and green, respectively. To distinguish more easily inter-session loop closures, Figure2.8 illustrates the global graph for y-value of the poses over time. Note that all paths for each session started at $y = 0$ and they were not connected together by neighbor links. Optimizing the graph using all these detected loop closures results in a single fully connected map of all five mapping sessions. Figure2.9 shows the

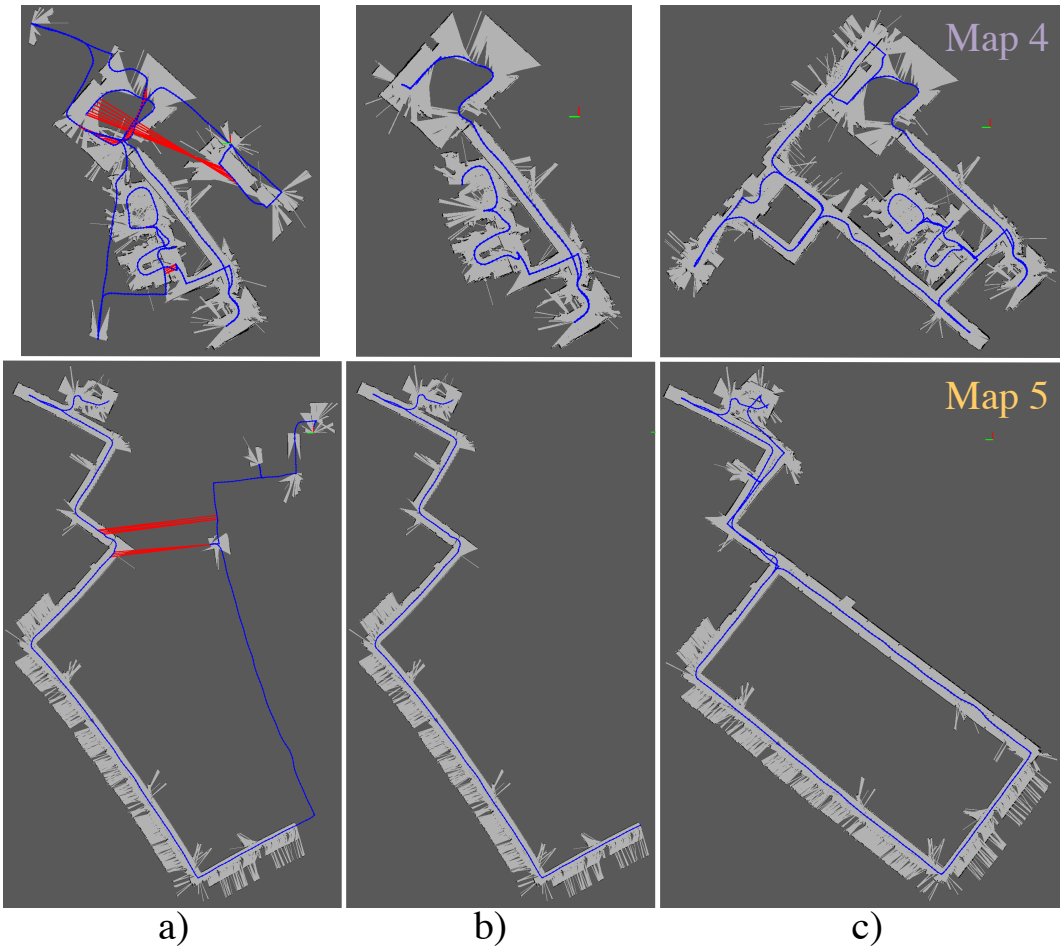


Figure 2.5 Results for Map 4 (top) and Map 5 (bottom), with a) the map from all nodes still in WM (light gray) with the global graph (blue line) not optimized, b) the local map with local graph optimization and c) the global map with global graph optimization. Loop closures are shown in red.

resulting global map by assembling the RGB-D point clouds from the Kinect using the optimized poses of the graph.

Figure2.10 a) shows the resulting local map created from all the mapping sessions. Because the local map is built only from nodes in WM that are linked (directly or indirectly) to the last node, only a small portion of the global map is available online. Note that the local map is also smaller than Map 5 taken independently (shown by Figure2.5): in the second experiment, there were nodes with more weight from previous mapping sessions that were still in WM, thus more nodes from the latest mapping session were transferred to LTM and not used for local map creation. These high weighted nodes are located in the light gray areas of Figure2.10 b). The blue line represents the global graph created using all constraints in LTM. When using all constraints in LTM, the local map is also

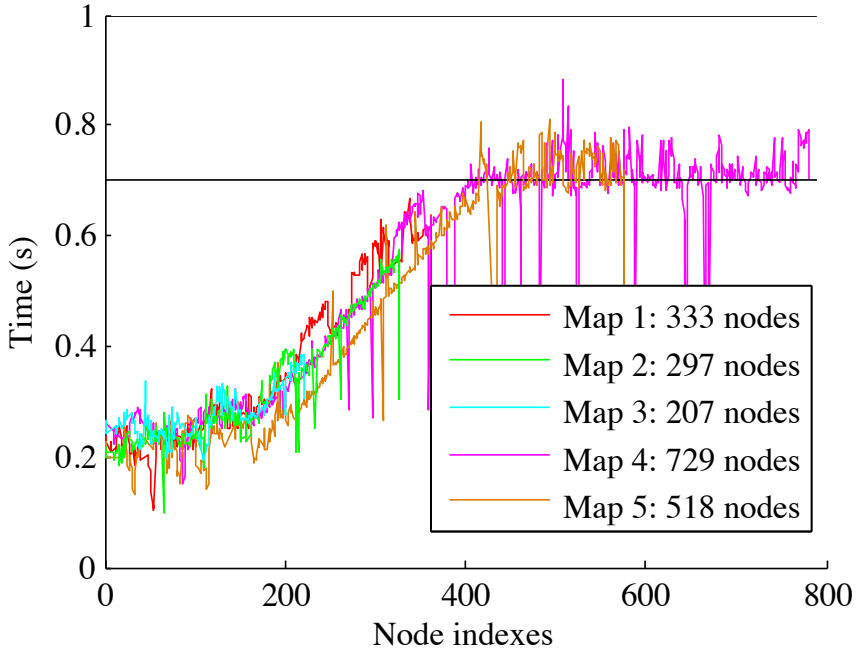


Figure 2.6 Processing time in relation to the number of nodes processed over time for each data set. T is shown by the horizontal line.

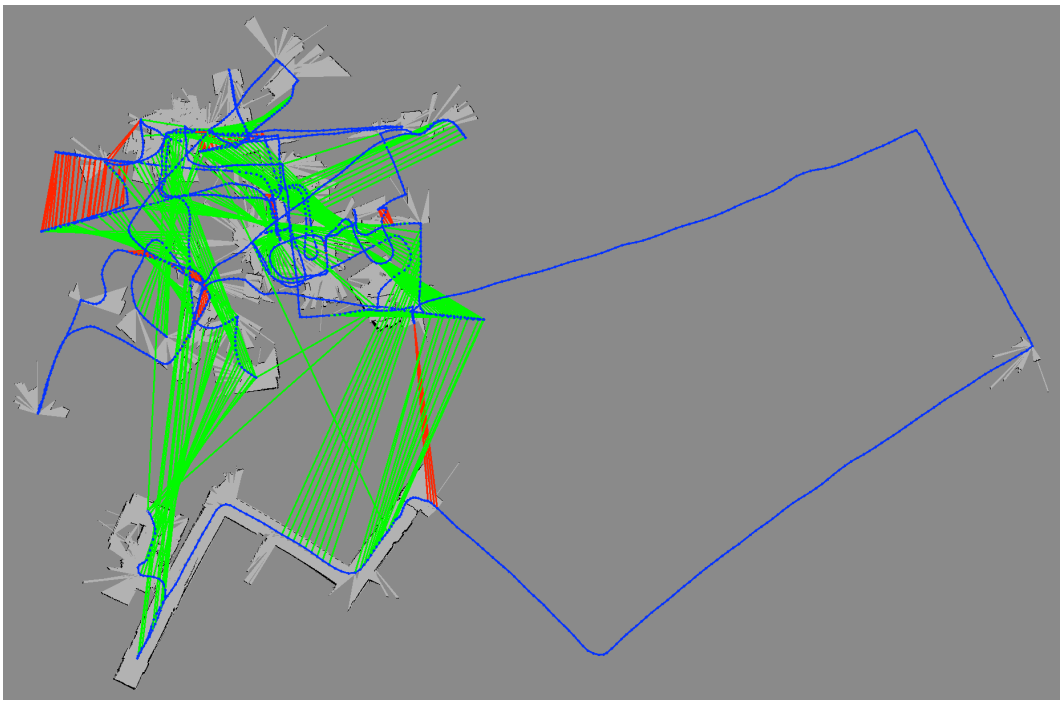


Figure 2.7 Top view of the map without optimization after five mapping sessions. The red and green links show intra-session and inter-session loop closures detected, respectively.

slightly more straight. At the end of the experiment, the global graph has 2074 nodes

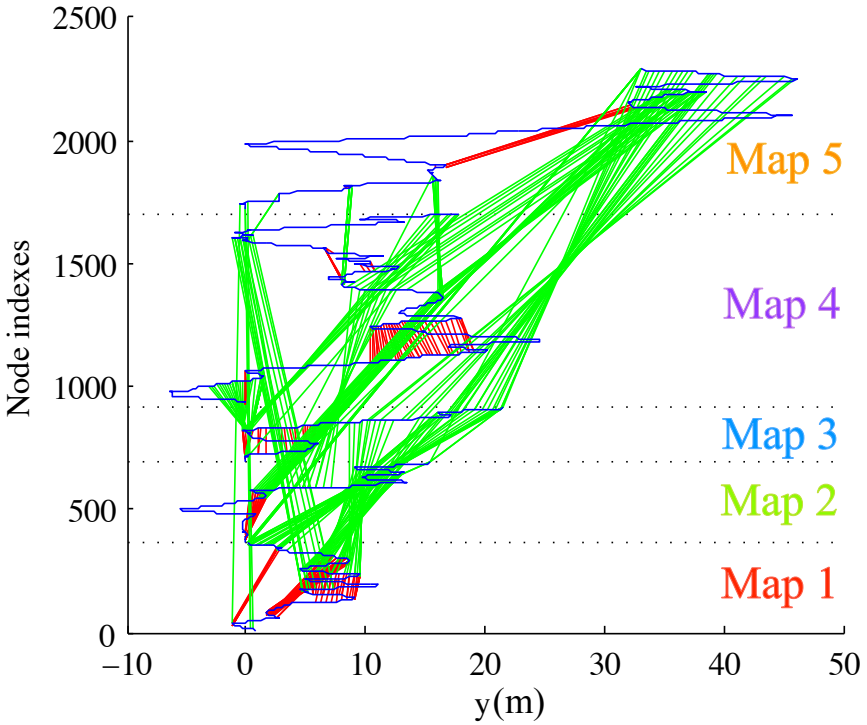


Figure 2.8 Loop closures between the mapping sessions. Only the y values of the poses are illustrated for visibility purposes. Green and red links are inter-session and intra-session loop closures detected, respectively. Neighbor links are shown in blue. Note that only green links connect the five maps together.

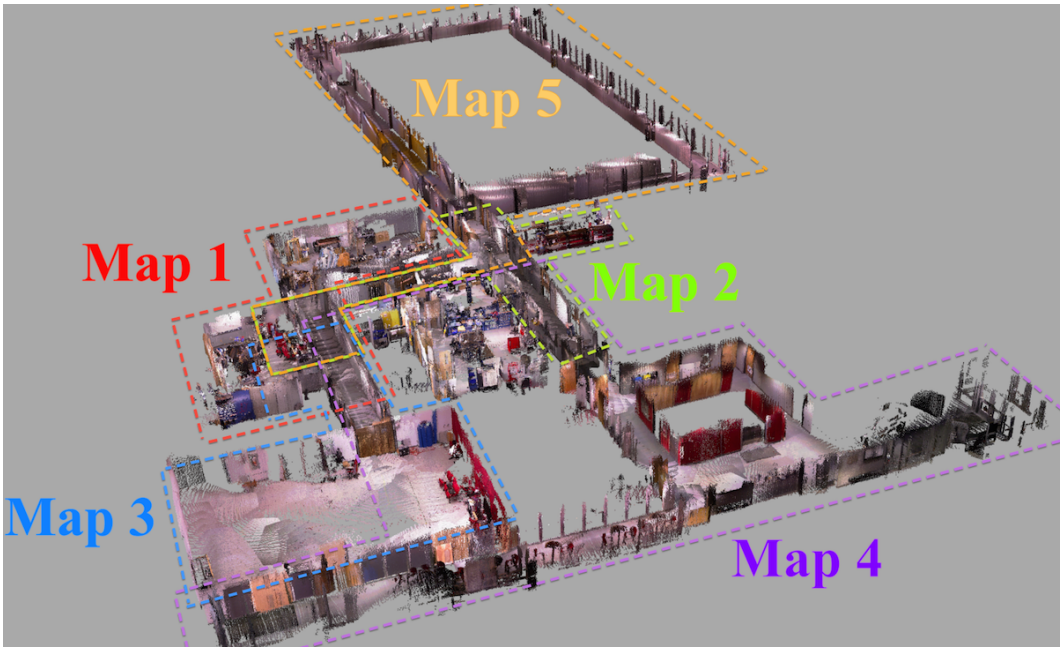


Figure 2.9 Five online mapping sessions merged together automatically.

with all mapping sessions connected, with 330 nodes in WM (107, 12, 27, 28, 156 nodes from maps 1, 2, 3, 4 and 5, respectively) for which 173 nodes are accessible for the local map (4, 15, 6, 0, 148 nodes from maps 1, 2, 3, 4 and 5, respectively). For the local map, it is normal that a high proportion of nodes are from the last session, which is the most recent one. Nodes from older maps are those retrieved from LTM around the latest loop closures found. For example, when the robot is mapping a new area, only nodes of the last session would be in the local map.

To observe the influence of memory management on the quality of the map created, we conducted the same experiment without T. All nodes were then kept in WM and they were processed by both loop closure detection and graph optimization at each time step. Normally, without transferring nodes to LTM, more loop closures would be detected, so more constraints would be used for graph optimization. As shown in Figure2.12, the processing time becomes greater than the acquisition time R, which is not the case with $T = 0.7$ s. However, without T, 193 intra-session and 387 inter-session loop closures were detected, comparatively to 188 and 258 respectively for the online experiment. Figure2.11 compares the resulting global maps with (blue) and without (red) T. By comparing with the building plan (the plan was scaled to 5 cm / pixel like the generated maps, the maps were manually oriented so trajectories are aligned to most doors traversed), the quality of the experiment without T (red) is a little better than with T (blue), probably because more loop closures were used for graph optimization. However, for the two conditions, the large loop from Map 5 is not correctly aligned with the building plan. The robot traversed this area only once and exited from the same door from which it entered, making it more difficult for the graph optimization algorithm to correct angular errors for this single entry point. For comparison, the left part of the map was also traversed once during session 4, but the robot exited the area from another door, thus making the area more robust to angular errors.

2.4 Discussion

In term of processing time, the results show that the proposed approach is able to satisfy online processing requirements independently of the size of the environment. However, map quality depends on the number of loop closures that can be detected. To satisfy online requirements, the robot transfers in LTM some portions of the map which cannot be used for loop closure detection. For multi-session mapping, the worst case would occur if all nodes of a previous map are transferred to LTM before a loop closure is detected with the new map. This would result in definitely forgetting the previous map: there

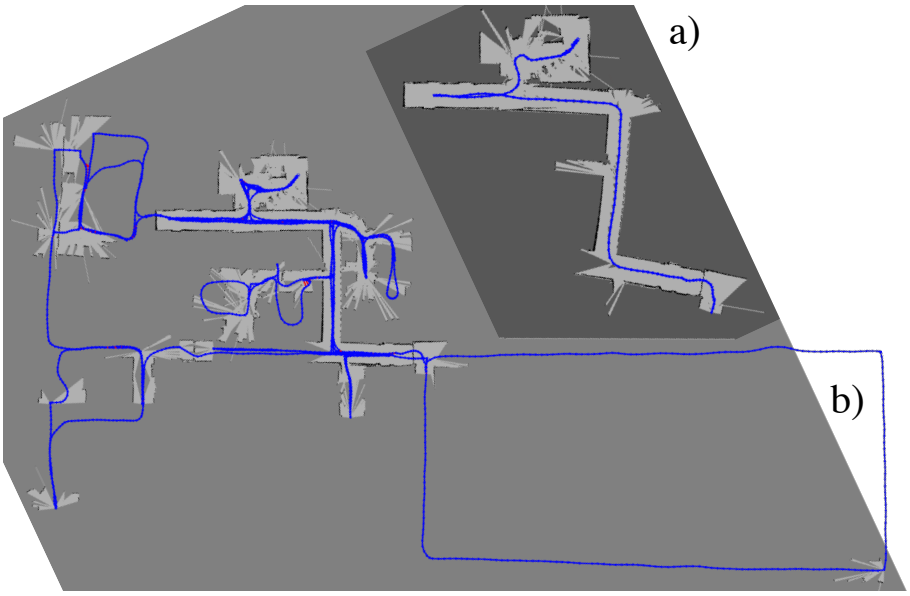


Figure 2.10 Graphs optimized for a) the last local map built online, b) the global map built offline, with nodes in light gray areas are those still in WM, and the other nodes are in LTM.

Figure 2.11 Global maps with (blue) and without (red) T. The maps are manually superimposed over the actual plan of the building.

would be no links in WM and even in LTM that could connect this older map to the new one, and it would be ignored even for the global map construction. To avoid this problem, our approach could keep at least one node for each map in WM. However, if the number

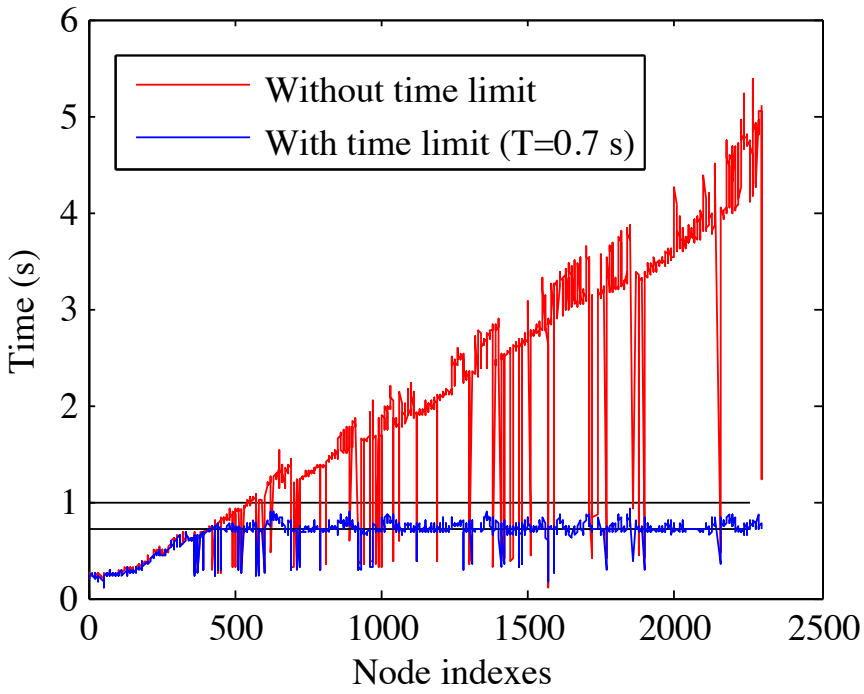


Figure 2.12 Processing time for each node added to graph. The horizontal lines are $T = 0.7$ and $R = 1$.

of mapping sessions becomes very high (e.g., thousands of sessions), these nodes would definitely have to be transferred in LTM to satisfy the online requirement. For long-term, large-scale and multi-session mapping, some portions of the map would then be definitely forgotten, and therefore some kind of heuristic to efficiently manage important nodes to keep in WM is required.

Another observation is that frequently revisiting old maps increases global map quality. A robot autonomously mapping a facility could, when detecting an old map, decide to revisit some parts of it to detect more inter-session loop closures, thus creating more constraints for graph optimization.

In the experiments conducted, no invalid loop closures were detected. If this occur, erroneous constraints would be added to graph optimization, resulting in map errors. Some graph optimization approaches such as [Latif *et coll.*, 2012; Sunderhauf et Protzel, 2012] deal with possible invalid matches, and could be used to increase robustness of the proposed approach.

2.5 Conclusion

Results presented in this paper suggest that the proposed graph-based SLAM approach is able to meet online requirements needed for large-scale, long-term and multi-session online mapping. By limiting the number of nodes in WM available for global loop closure detection and graph optimization, online processing is achieved for new data acquired. Our approach is tightly based on global loop closure detection, allowing it to naturally deal with the kidnapped robot problem and gross errors in odometry. Our code is open source and available at `http://introlab.github.io/rtabmap/`. In future work, we plan to study the impact of autonomous exploration strategies on multi-session mapping, especially how it can actively direct exploration based on nodes available for online mapping and graph optimization.

CHAPITRE 3

Long-Term Online Multi-Session Graph-Based SPLAM with Memory Management

Avant-propos

Auteurs et affiliations :

M. Labbé : étudiant au doctorat, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique.

F. Michaud : professeur, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique.

Date d'acceptation : 6 novembre 2017

État de l'acceptation : version finale publiée en août 2018, volume 42, numéro 6 (`https://doi.org/10.1007/s10514-017-9682-5`).

Revue : *Autonomous Robots*

Référence : [Labbé et Michaud, 2017]

Titre français : Planification, localisation et cartographie simultanées 'en ligne' et à long terme avec gestion de mémoire

Contribution au document : Cet article contribue à la thèse en élaborant comment la gestion de mémoire peut être intégrée à un robot faisant continuellement du SLAM tout en navigant de façon autonome. Le défi est d'avoir assez d'information dans la mémoire de travail du robot pour correctement suivre une trajectoire planifiée tout en étant capable de se relocaliser globalement sur une longue période d'opération, et ce, toujours en effectuant le traitement 'en ligne'.

Résumé français : Pour la planification, la localisation et la cartographie simultanées à long terme (SPLAM), un robot doit pouvoir mettre continuellement à jour sa carte en fonction des changements dynamiques de l'environnement et des nouvelles zones explorées. Avec des capacités de calcul embarquées limitées, un robot doit également pouvoir limiter la taille de la carte utilisée pour que la localisation et la cartographie soient toujours possible

par du traitement ‘en ligne’. Cet article aborde ces défis en utilisant un mécanisme de gestion de mémoire qui identifie les endroits qui doivent rester dans une mémoire de travail (WM) pour être traités ‘en ligne’, et à ceux qui doivent être transférés vers une mémoire à long terme (LTM). Lorsque des endroits précédemment transférés dans la LTM sont revisités, le mécanisme de gestion de mémoire peut récupérer ces endroits et les replacer dans la WM pour être utilisés par le SPLAM. L’approche est testée sur un robot équipé d’un télémètre laser à courte portée et d’une caméra RGB-D, patrouillant de manière autonome un total de 10,5 km dans un environnement intérieur sur 11 sessions tout en ayant rencontré 139 personnes.

Abstract

For long-term simultaneous planning, localization and mapping (SPLAM), a robot should be able to continuously update its map according to the dynamic changes of the environment and the new areas explored. With limited onboard computation capabilities, a robot should also be able to limit the size of the map used for online localization and mapping. This paper addresses these challenges using a memory management mechanism, which identifies locations that should remain in a Working Memory (WM) for online processing from locations that should be transferred to a Long-Term Memory (LTM). When revisiting previously mapped areas that are in LTM, the mechanism can retrieve these locations and place them back in WM for online SPLAM. The approach is tested on a robot equipped with a short-range laser rangefinder and a RGB-D camera, patrolling autonomously 10.5 km in an indoor environment over 11 sessions while having encountered 139 people.

3.1 Introduction

The ability to simultaneously map an environment, localize itself in it, and plan paths using this information is known as *Simultaneous Planning, Localization And Mapping*, or SPLAM [Stachniss, 2009]. This task can be particularly complex when done online on a robot with limited computing resources in large, unstructured and dynamic environments. Since SPLAM can be seen as an extension of *Simultaneous Localization And Mapping* (SLAM), many approaches exist [Thrun *et coll.*, 2005]. Our interest lies with graph-based SLAM approaches [Grisetti *et coll.*, 2010], for which combining a lightweight topological map over a detailed metrical map reveals to be more suitable for large-scale mapping and navigation [Konolige *et coll.*, 2011].

Two important challenges in graph-based SPLAM are :

– Multi-session mapping, also known as the *kidnapped robot problem* or the *initial state problem*: when turned on, a robot does not know its relative position to a map previously created, making it impossible to plan a path to a previously visited location. A solution is to have the robot localize itself in a previously-built map before initiating mapping. This solution has the advantage of always using the same referential, resulting in only one map is created across the sessions. However, the robot must start in a portion already mapped of the environment. Another approach is to initialize a new map with its own referential on startup, and when a previously visited location is encountered, a transformation between the two maps can be computed. The transformations between the maps can be saved explicitly

with special nodes called *anchor nodes* [Kim *et coll.*, 2010; McDonald *et coll.*, 2012], or implicitly with links added between each map [Konolige et Bowman, 2009; Latif *et coll.*, 2013]. This process is referred to as *loop closure detection*. Loop closure detection approaches that are independent of the robot's estimated position [Ho et Newman, 2006] can intrinsically detect if the current location is a new location or a previously visited one among all the mapping sessions conducted in the past. Popular loop closure detection approaches are appearance-based [Garcia-Fidalgo et Ortiz, 2015], exploiting the distinctiveness of images of the environment. The underlying idea is that loop closure detection is done by comparing all previous images with the new one. When loop closures are found between the maps, a global map can be created by combining the maps from each session. In graph-based SLAM, graph pose optimization approaches [Folkesson et Christensen, 2007; Grisetti *et coll.*, 2007a; Johannsson *et coll.*, 2013; Kummerle *et coll.*, 2011] use these loop closures to reduce odometry errors inside each map and in between the maps.

– Long-term mapping in dynamic environments. *Persistent* [Milford et Wyeth, 2010], *lifelong* [Konolige et Bowman, 2009] or *continuous* [Pirker *et coll.*, 2011] are terms generally used to describe SLAM approaches working in such conditions. Continuously updating and adding new data to the map in unbounded or dynamic environments will inevitably increase the map size over time. Online simultaneous planning, localization and mapping requires that new incoming data be processed faster than the time to acquire them. For example, if data are acquired at 1 Hz, updating the map should be done in less than 1 sec. As the map grows, the time required for loop closure detection and graph optimization increases, and eventually limits the size of the environment that can be mapped and used online.

To address these challenges, we introduce SPLAM-MM, a graph-based SPLAM with a memory management (MM) mechanism. As demonstrated in [Labbé et Michaud, 2013], memory management can be used to limit the size of the map so that loop closure detections are always processed under a fixed time limit, thus satisfying online requirements for long-term and large-scale environment mapping. The idea behind SPLAM-MM is to limit the number of nodes available for loop closure detection and graph optimization, keeping enough observations in the map for successful online localization and planning while still having the ability to generate a global representation of the environment that can adapt to changes over time.

The paper is organized as follows. Section 3.2 reviews graph-based SLAM approaches that reduce the size of the map when revisiting the same environment while continuously

adapting to dynamic changes. Section 3.3 describes the implementation and the operating principles associated with the use of memory management with a graph-based SPLAM approach, which extends our previous metric-based SLAM approach [Labbé et Michaud, 2014] with a new planning capability. The implementation integrates four algorithms: loop closure detection [Labbé et Michaud, 2013], graph optimization [Grisetti *et coll.*, 2007a], metrical path planner [Marder-Eppstein *et coll.*, 2010] and a custom topological path planner. Section 3.4 presents experimental results of 11 SPLAM sessions using the AZIMUT-3 robot in an indoor environment over 10.5 km. Section 3.5 discusses strengths and limitations of SPLAM-MM, and Section 3.6 concludes the paper.

3.2 Related Work

Lifelong appearance-based SLAM requires dealing with dynamic environments. [Glover *et coll.*, 2010] present an appearance-based SLAM approach that had to operate in different lighting conditions over three weeks. An interesting observation from their experiments is that even when revisiting the same locations, the map still grows: in dynamic environments, the loop closure detector is sometimes unable to detect loop closures, duplicating locations in the map. A map management approach is therefore required to limit map size. In highly dynamic environments, multiple views of the same location may also be required for proper localization. [Churchill et Newman, 2012] present a graph-based SLAM approach where visual experiences of the same locations are kept in the map, to increase localization robustness to dynamic changes caused for instance by outdoor illumination conditions. If localization fails when revisiting an area, new experiences are added to the map. Even if adding new visual experiences to the map happens less often over time (as the robot explores the same location), there is no mechanism to limit this. [Pirker *et coll.*, 2011] present a continuous monocular SLAM approach where new key frames are added to the map only when the environment has changed, to keep its size proportional to the explored space. But if the environment changes very often, there is no mechanism to limit the number of key frames over the same physical location.

Some SLAM approaches can handle dynamic changes of the environment while limiting the size of the map for long-term operation. [Biber et Duckett, 2005] present a sample-based representation for maps, to handle changes at different timescales, tracking both stationary and non-stationary elements of the environment. The idea is to refresh samples stored for each timescale with new sensor measurements. Map growth is then indirectly limited as older memories fade at different rates depending on the timescale. [Walcott-Bryant *et coll.*, 2012] describe Dynamic Pose-Graph SLAM (DPG-SLAM), a long-term mapping

approach that detects static and dynamic changes of the environment through time. To keep consistency of the graph while reducing its size, nodes that are not observable anymore are removed. [Johannsson *et coll.*, 2013] also remove unobservable nodes to limit the size of the map over time when revisiting the same area. Similar nodes of the graph are merged together while keeping only the new loop closure detection. However, the graph size is not bounded when exploring new areas. [Krajník *et coll.*, 2016] present an occupancy grid approach where each cell in the map estimates its occupancy value depending on periodical and cyclic changes occurring in the environment. This increases localization and navigation accuracy in dynamic environments compared to static maps, as the predicted map represents the correct state of the environment at that time of the day (e.g., doors can change to be opened or closed). The maximum data kept for each cell is bounded by some parameters (depending on the smallest and longest cyclic periods that should be detected), thus keeping memory usage fixed. However, the approach assumes that the navigation phase always occur in the same environment as the first mapping cycle, without possibility to extend it afterward.

These problems of lifelong SLAM are also addressed in some SPLAM approaches. [Milford et Wyeth, 2010] present a solution to limit the size of the map (called experience map) while revisiting the same area: close nodes are merged together up to a maximum density threshold. This approach has the advantage of making the map size independent of the operating time, but the diversity of the observations on each location is somewhat lost. [Konolige *et coll.*, 2011] use a view-based graph SLAM approach [Konolige et Bowman, 2009] in a SPLAM context. The approach preserves diversity of the images referring to the same location so that the map can handle dynamic changes over time, and forgetting images limits the size of the graph over time when revisiting the same area. However, the graph still grows when visiting new areas.

Overall, these approaches reduce map size when revisiting the same area, while continuously adapting to dynamic changes. This makes them independent or almost independent of the operation time of the robot in these conditions, but they are all limited to a maximum size of the environment that can be mapped online. The SPLAM-MM approach deals specifically with this limitation.

3.3 Memory Management for SPLAM

The underlying representation of SPLAM-MM is a graph with nodes and links. The nodes contain the following information:

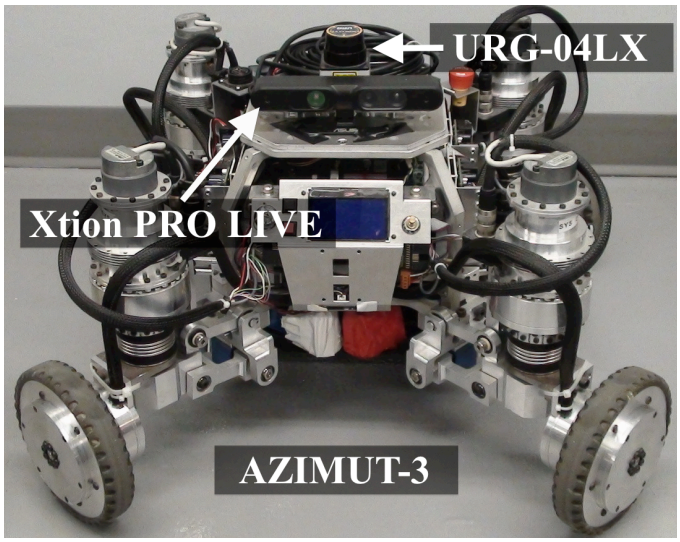


Figure 3.1 The AZIMUT-3 robot equipped with a URG-04LX laser range finder and a Xtion PRO LIVE sensor.

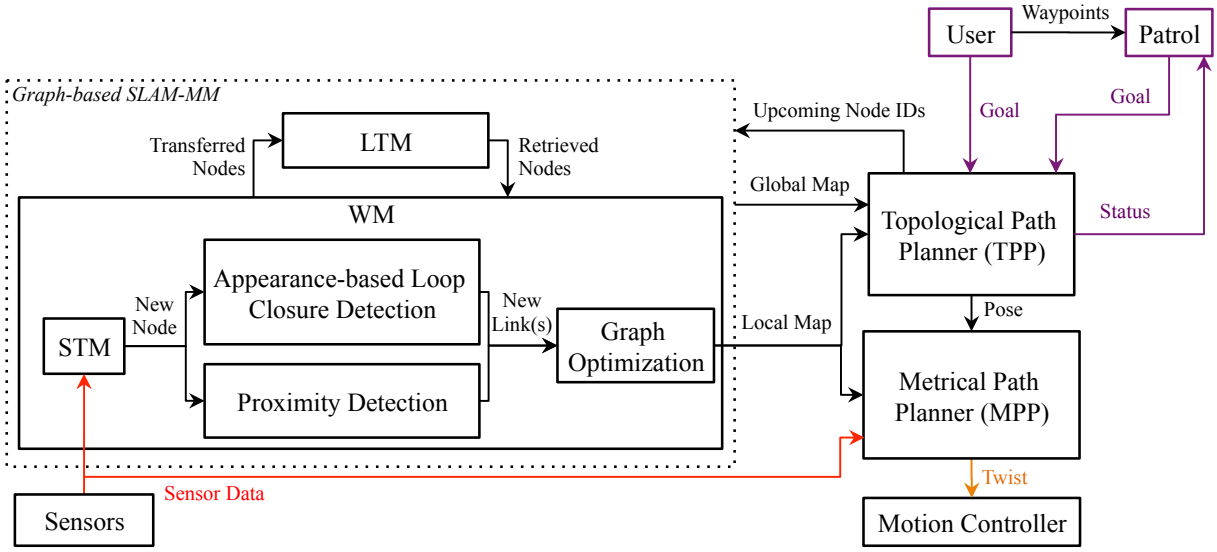


Figure 3.2 Memory management and control architecture of SPLAM-MM.

- ID: unique time index of the node.
- Weight: an indication of the importance of the node, used for memory management.
- Bag-of-words (BOW): visual words used for loop closure detections. They are SURF features [Bay *et coll.*, 2008] quantized to an incremental vocabulary based on KD-Trees.
- Sensor data: used to find similarities between nodes and to construct maps. For this paper, our implementation of SPLAM-MM is using the AZIMUT-3 robot [Ferland *et coll.*, 2010], equipped with an URG-04LX laser rangefinder and a Xtion Pro Live RGB-D camera, as shown by Figure 3.1. The sensory data used are:

- Pose: the position of the robot computed by its odometry system (e.g., the value given by wheel odometry), expressed in (x, y, θ) coordinates.
- RGB image: used to extract visual words.
- Depth image: used to find 3D position of the visual words. The depth image is registered with the RGB image, i.e., each depth pixel corresponds exactly to the same RGB pixel.
- Laser scan: used for loop closure transformations and odometry refinements, and by the Proximity Detection module.

The links store rigid transformations (i.e., Eucledian transformation derived from odometry or loop closures) between nodes. There are four types of links:

- Neighbor link: created between a new node and the previous one.
- Loop closure link: added when a loop closure is detected between the new node and one in the map.
- Proximity link: added when two close nodes are aligned together.
- Temporary link: used for path planning purposes. It is used to keep the planned path connected to the current map.

Figure 3.2 presents a high-level representation of SPLAM-MM. Basically, it consists of a graph-based SLAM module with memory management, to which path planners are added. Memory management involves the use of a Working Memory (WM) and a Long-Term Memory (LTM). WM is where maps, which are graphs of nodes and links, are processed. To satisfy online constraints, nodes can be transferred and retrieved from LTM. More specifically, the WM size indirectly depends on a fixed time limit T: when the time required to update the map (i.e., the time required to execute the processes in the Graph-based SLAM-MM block) reaches T, some nodes of the map are transferred from WM to LTM, thus keeping WM size nearly constant and processing time around T. However, when a loop closure is detected, neighbors in LTM with the loop closure node can be retrieved from LTM to WM for further loop closure detections. In other words, when a robot revisits an area which was previously transferred to LTM, it can incrementally retrieve the area if a least one node of this area is still in WM. When some LTM nodes are retrieved, nodes in WM from other areas in the map can be transferred to LTM, to limit map size in WM and therefore keeping processing time around T.

Therefore, the choice of which nodes to keep in WM is key in SPLAM-MM. The objective is to have enough nodes in WM from each mapping session for loop closure detections and to keep a maximum number of nodes in WM for generating a map usable to follow

correctly a planned path, while still satisfying online processing. Two heuristics are used to establish the compromise between selection of which nodes to keep in WM and online processing:

- Heuristic 1 is inspired from observations made by psychologists [Atkinson et Shiffrin, 1968; Baddeley, 1997] that people remember more the areas where they spent most of their time, compared to those where they spent less time. In terms of memory management, this means that the longer the robot is at a particular location, the larger the weight of the corresponding node should be. Oldest and less weighted nodes in WM are transferred to LTM before the others, thus keeping in WM only the nodes seen for longer periods of time. As demonstrated in [Labbé et Michaud, 2013], this heuristic reveals to be quite efficient in establishing the compromise between search time and space, as driven by the environment and the experiences of the robot.
- Heuristic 2 is used to identifies nodes that should stay in WM for autonomous navigation. Nodes on a planned path could have small weights and may be identified for transfer to LTM by Heuristic 1, thus eliminating the possibility of finding a loop closure link or a proximity link with these nodes and correctly follow the path. Therefore, Heuristic 2 must supersede Heuristic 1 and allow upcoming nodes to remain in WM, even if they are old and have a small weight.

The Graph-based SLAM-MM block provides two types of maps derived from nodes in WM and LTM:

- Local map, i.e., the largest connected graph that can be created from the last node in WM with nodes available in WM only. The local map is used for online path planning.
- Global map, i.e., the largest connected graph that can be created from the last node in WM with nodes in WM and LTM. It is used for offline path planning.

Figure 3.3 uses diamonds to represent initial and end nodes for each mapping session. The nodes in LTM are shown in red and the others are those in WM. The local map is created using only the nodes in WM that are linked to the last node. The graph linking the last node with other nodes in WM and LTM represents the global map (outer dotted area). If loop closure detections are found between nodes of different maps, loop closure links can be generated, and the local map can span over multiple mapping sessions. Other nodes in WM but not included in the local map are unreachable from the last node, but they are

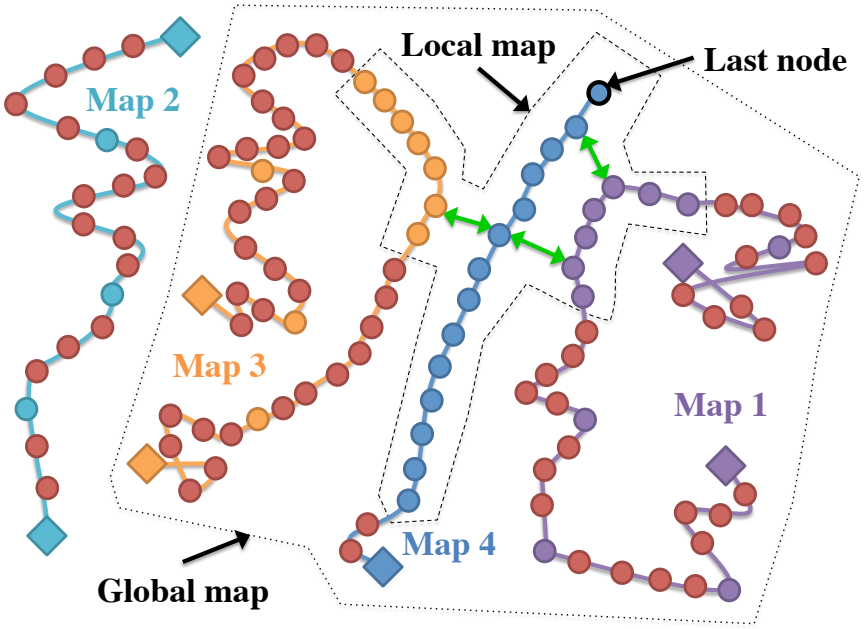


Figure 3.3 Illustration of the local map (inner dashed area) and the global map (outer dotter area) in multi-session mapping. Red nodes are in LTM, while all other nodes are in WM. Loop closure links are shown using bidirectional green arrows.

still used for loop closure detections since all nodes in WM (including those in Map 2 for instance) are examined.

The modules presented in Figure 3.2 are described as follows.

3.3.1 Short-Term Memory Module

Short-Term Memory (STM) is the entry point where sensor data are assembled into a node to be added to the map. Similarly to [Labbé et Michaud, 2013], the role of the STM module is to update node weight based on visual similarity. When a node is created, a unique time index ID is assigned and its weight is initialized to 0. The current pose, RBG image, depth image and laser scan readings are also memorized in the node. If two consecutive nodes have similar images, i.e., the ratio of corresponding visual words between the nodes is over a specified threshold Y, the weight of the previous node is increased by one. If the robot is not moving (i.e., odometry poses are the same), the new node is deleted. To reduce odometry errors on successive STM nodes, transformation refinement is done using 2D iterative-closest-point (ICP) optimization [Besl et McKay, 1992] on the rigid transformation of the neighbor link with the previous node and the corresponding laser scans. If the ratio of ICP point correspondences between the laser scans over the

total laser scan size is greater or equal to C, the neighbor link's transformation is updated with the correction.

When the STM size reaches a fixed size limit of S nodes, the oldest node in STM is moved to WM. STM size is determined based on the velocity of the robot and at which rate the nodes are added to the map. Images are generally very similar to the newly added node, keeping S nodes in STM avoids using them for appearance-based loop closure detection once in WM. For example, at the same velocity, STM size should be larger if the rate at which the nodes are added to map increases, in order to keep nodes with consecutive similar images in STM. Transferring nodes with images very similar with the current node from STM to WM too early limits the ability to detect loop closures with older nodes in WM.

3.3.2 Appearance-based Loop Closure Detection Module

Appearance-based loop closure detection is based on the bag-of-words approach described in [Labbé et Michaud, 2013]. Briefly, this approach uses a bayesian filter to evaluate appearance-based loop closure hypotheses over all previous images in WM. When a loop closure hypothesis reaches a pre-defined threshold H, a loop closure is detected. Visual words of the nodes are used to compute the likelihood required by the filter. In this work, the Term Frequency-Inverse Document Frequency (TF-IDF) approach [Sivic et Zisserman, 2003] is used for fast likelihood estimation, and FLANN (Fast Library for Approximate Nearest Neighbors) incremental KD-Trees [Muja et Lowe, 2009] are used to avoid rebuilding the vocabulary at each iteration. To keep it balanced, the vocabulary is rebuilt only when it doubles in size.

The RGB image, from which the visual words are extracted, is registered with a depth image. Using (3.1), for each 2D point (x, y) in the rectified RGB image, a 3D position P_{xyz} can be computed using the calibration matrix (focal lengths f_x and f_y, optical centres c_x and c_y) and the depth information d for the corresponding pixel in the depth image. The 3D positions of the visual words are then known. When a loop closure is detected, the rigid transformation between the matching images is computed using a RANSAC (RANdom SAmple Consensus) approach which exploits the 3D visual word correspondences [Rusu et Cousins, 2011]. If a minimum of I inliers are found, the transformation is refined using the laser scans in the same way as the odometry correction in STM using 2D ICP transformation refinement. If transformation refinement is accepted, then a loop closure link is added with the computed transformation between the corresponding nodes. The weight of the current node is updated by adding the weight of the loop closure hypothesis

node and the latter is reset to 0, so that only one node with a large weight represents the same location.

$$P_{xyz} = \left[\frac{(x - c_x) \cdot d}{f_x}, \frac{(y - c_y) \cdot d}{f_y}, d \right]^T \tag{3.1}$$

By doing appearance-based loop closure detection this way, setting H high means that there is less chance of detecting false positives, but at the cost of detecting less loop closures [Labbé et Michaud, 2013]. For SPLAM-MM, H can be set relatively low to detect more loop closures because false positives that are geometrically different will be rejected by the rigid transformation computation step (i.e., the 3D visual word correspondences and 2D ICP transformation refinement).

3.3.3 Proximity Detection Module

Appearance-based loop closure detection is limited by the perceptual range of the sensory data used. For instance, when the robot is revisiting areas in opposite direction, the RGB-D camera on AZIMUT-3 is not pointing in the same direction compared to when the nodes were created, and thus no appearance-based loop closures can be detected. This also happens when there are not enough visual features under the depth range of the RGB-D camera (e.g., white walls or long halls). Simply relying on appearance-based loop closure detections for map corrections would then limit path planning capabilities, and make navigation difficult in such conditions. Figure 3.4a illustrates a situation where the robot is in a hall coming back to its starting position in reverse direction. Setting a goal at the starting position would make the planner fail because no loop closures could be found to correct the odometry, resulting in having a wall directly placed on the starting position. One solution would be to have the robot visit the nodes of the graph backward so loop closures could be detected to correct the map, and ultimately be able to reach the starting position. However, it is inefficient and unsafe if the robot does not have sensors pointing backward. To deal with such situations, the Proximity Detection module uses laser rangefinder data to correct odometry drift in areas where the camera cannot detect loop closures. With a field of view of more than 180°, the laser scans can be aligned in reverse direction, generating proximity links. As laser scans are not as discriminative as images, proximity detection is restricted to nodes of the local map located around the estimated position of the robot. Figure 3.4b illustrates the result.

Figure 3.5 illustrates how nodes located close to the robot are selected by the Proximity Detection module. Only nodes in the local map with their pose inside radius R centered

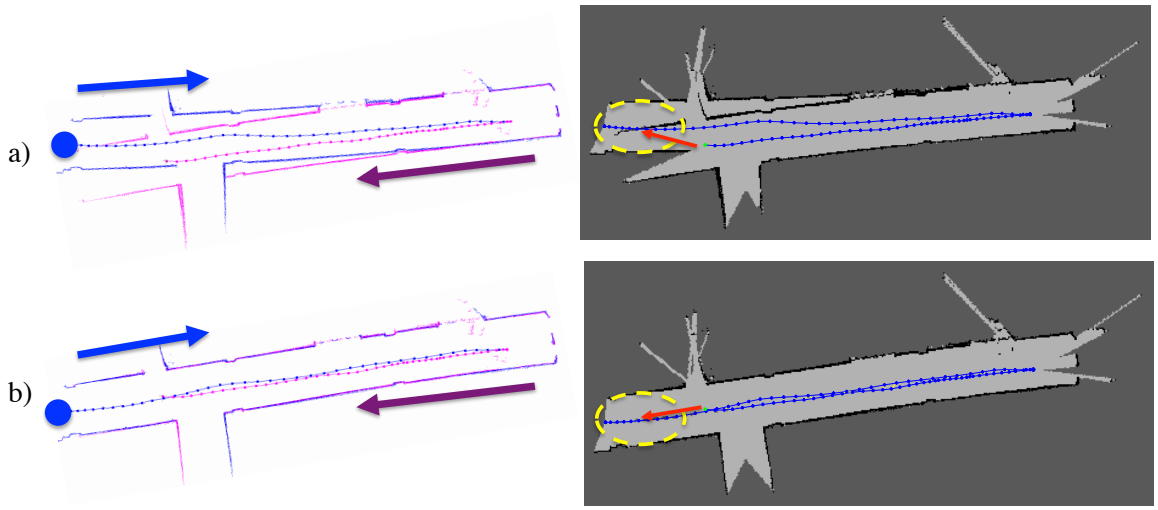


Figure 3.4 Illustration of the role of the Proximity Detection module. On the left are the raw laser scans, the blue dot is the starting position, and on the right the corresponding occupancy grid map at 0.05 m resolution (black, light gray and dark gray areas are occupied, empty and unknown spaces, respectively). In a), the yellow circle on the right locates the problematic situation: after the second traversal, the first nodes of the graph are located exactly over the wall, making it impossible to plan a path (red arrow on the right) to return to the starting position. In b), proximity links are detected using only the laser scans, and the local map can then be correctly optimized.

on the robot are used. Nodes in STM are not considered in order to avoid adding useless links with nodes close by: this would increase graph optimization time without adding significative improvements of the map. The nodes are then segmented into groups with nodes connected only by neighbor links. A group must have its nearest node from the robot inside a fixed radius L defining close-by nodes (with $L < R$) to be considered for proximity detection, to keep the length of the resulting proximity links small for path planning. Note that Appearance-based Loop Closure Detection is done before Proximity Detection, thus if the nearest node has already a loop closure with the new node, the group is ignored. Proximity detection is then applied separately on each group of nodes by doing the following steps:

1. A rigid transformation between the nearest node of each group and the new node added to map is computed as in Section 3.3.2, and if it is accepted, a proximity link is added between the corresponding nodes, and the group of nodes is ignored for step 2. These links are referred as visual proximity links because visual words are used in the transformation estimation.

2. To avoid having to compare multiple nodes with very similar laser scans (and thus to save computation), only the more recent node among those in the same fixed small radius L (centered on each node) is kept along the nodes in a remaining group. Then for each group, the laser scans of the nodes are merged together using their respective pose. 2D ICP transformation refinement is done between the merged laser scans and the one of the new node. If the transformation is accepted, a new proximity link with this transformation is added to the graph between the new node and the nearest one in the group.

3.3.4 Graph Optimization Module

TORO (Tree-based netwORk Optimizer) [Grisetti *et coll.*, 2007a] is used for graph optimization. When loop closure and proximity links are added, the errors derived from odometry can be propagated to all links, thus correcting the local map. This also guarantees that nodes belonging to different maps are transformed into the same referential when loop closures are found.

When only one map exists, it is relatively straightforward to use TORO to create a tree because it only has one root. However, for multi-session mapping, each map has it own root with its own reference frame. When loop closures occur between the maps, TORO cannot optimize the graph if there are multiple roots. It may also be difficult to find a unique root when some of the nodes have been transferred in LTM. As a solution, our approach takes the root of the tree to be the latest node added to the local map, which is always uniquely defined across intra-session and inter-session mapping. All other poses in the graph are then optimized using the last odometry pose as the referential.

3.3.5 Path Planning Modules

Memory management has a significant effect on how to do path planning online using graph-based SLAM, for which the map changes almost at each iteration and with only the local map accessible while executing the plan. This differs from approaches that assume that the map is static and/or that all the previously visited locations always remain in the map. In this paper, SPLAM-MM uses two path planners: a Metrical Path Planner (MPP) and a Topological Path Planner (TPP).

Metrical Path Planning Module

MPP receives a pose expressed in (x, y, θ) coordinates, and uses the local map to plan a trajectory and to make the robot move toward the targeted pose while avoiding obstacles.

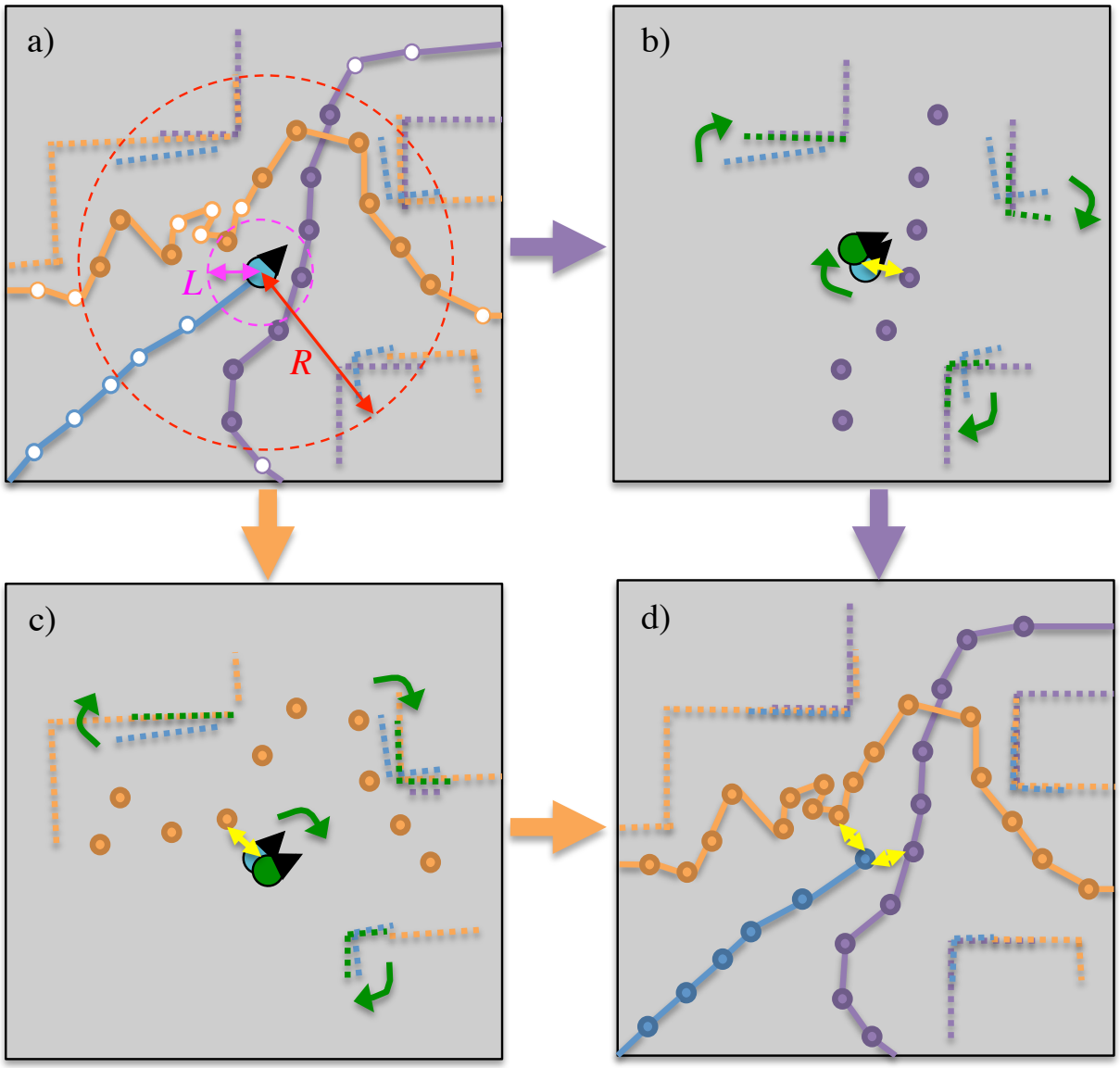


Figure 3.5 Illustration of how proximity detection works. In a), the larger dashed circle represents the radius R used to determine close-by nodes, and the smaller dashed circle defined by L is used to limit the length of the links to be created. The empty dots are nodes for which the laser scans are not used, either because they are outside the radius R, they are too close from each other or they are in STM. In b) and c), nodes in the radius R from the two segmented groups of nodes are processed for proximity detection. In d), proximity links are added (yellow), and after graph optimization, the groups of nodes are connected together and the respective laser scans are now aligned.

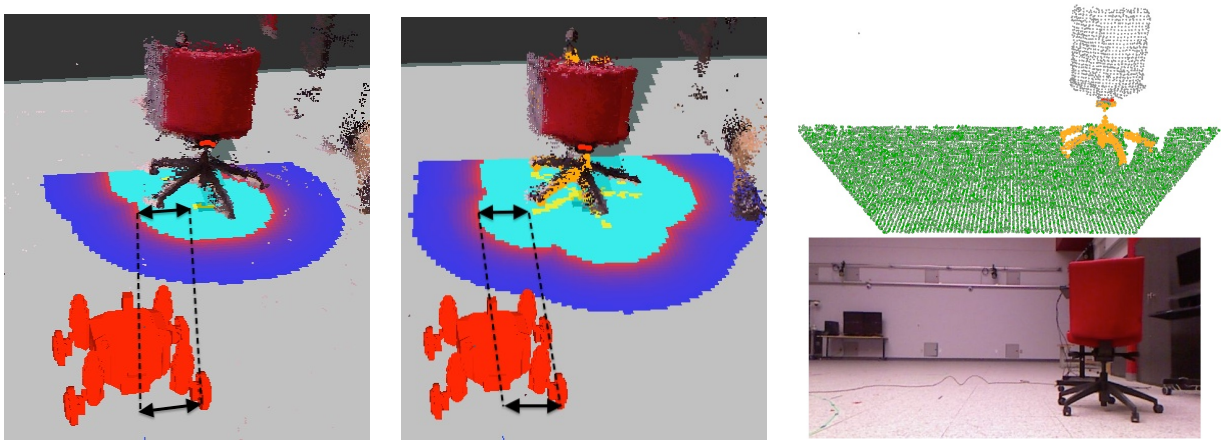
Our MPP implementation exploits the ROS navigation stack [Marder-Eppstein *et coll.*, 2010] to compute trajectories expressed as a sequence of velocity commands (expressed as twists) sent to the robot's Motion Controller module. A global Costmap is used to plan a

trajectory to a targeted pose. MPP creates the global Costmap from an occupancy grid created using the assembled laser scans from the latest local map. Each time the local map is updated, the occupancy grid is re-assembled and the trajectory is re-planned. MPP also uses a local Costmap for its Dynamic Window Approach (DWA) [Fox *et coll.*, 1997] to handle dynamic obstacles for collision avoidance. The local Costmap is created directly from sensor readings. To create the local Costmap, only using the laser rangefinder for obstacle detection revealed to be insufficient: while the laser range finder can detect most of the obstacles (e.g., walls, people, table legs), it is located 40 cm above the floor and all obstacles under this height cannot be detected. Therefore, the depth image from the RGB-D camera is also used to detect these small obstacles and to add them to the local Costmap. Figure 3.6 shows an example where combining laser scans and RGB-D data creates a more robust and a safer local Costmap for navigation. Note that segmentation of the point cloud generated from the depth image is required to be able to add or clear small dynamic obstacles below the RGB-D camera. To segment the ground, all points with normal parallel to z-axis (up to an angle Z) are labeled as ground. Then, all other points under a maximum height U are labeled as obstacles. This method would also make the robot capable of operating on uneven terrain.

Topological Path Planning Module

When TPP receives a goal identified by a node ID from a user (or a high-level module like a task planner, or in this paper the Patrol module), the global map is provided by the graph-based SLAM-MM module, and a topological path is computed to reach this goal. The topological path is a sequence of poses, expressed by their respective node IDs, to reach the goal. This step must be done offline or when the robot is not moving because all nodes linked to the current local map should be retrieved from LTM to build the global map.

To choose which nodes to use for navigation, TPP computes a path from the current node to the goal node using Djikstra algorithm [Dijkstra, 1959]. The choice of using Dijkstra over A* is to avoid global graph optimization, which is time consuming, to know the distance to goal required by A*. Dijkstra can also be computed directly when fetching the global map from LTM. Similar to [Valencia *et coll.*, 2013], to avoid losing track of the planned path, TPP prefers paths traversed in the same direction (e.g., where the camera is facing the same direction than on the nodes on the path) over shortest paths. This increases localization confidence: loop closure detection and visual proximity detection are more reliable than proximity detection using only laser scans because of their double verification (3D visual word correspondences and 2D ICP transformation refinement). To



(a) (b) (c)

Figure 3.6 Example of obstacle detection using the laser rangefinder and the RGB-D camera. The red dots on the chair show what is detected using the laser rangefinder data. The cyan area is derived from the obstacle projection on the ground plane up to robot's footprint radius, delimiting where the center of the robot should not enter to avoid collisions. In a), only the laser rangefinder data are used and the chair's wheels are not detected, making unsafe for the robot to plan a path around the chair. In b), the point cloud generated from the camera's depth image is used and the chair's wheels are detected (shown by the orange dots), increasing the cyan area (and consequently the area to avoid colliding with the chair). Illustration c) presents a view from the RGB-D camera where the segmented ground is shown in green and the obstacles in orange.

embed this preference in Djikstra, the search cost is angular-based instead of distance-based, i.e., it finds the path with less orientation changes when traversing it in the forward direction.

Then, TPP selects the farthest node on the path in the local map and sends its pose to MPP. While MPP makes the robot navigate to its targeted pose, TPP indicates to the graph-based SLAM-MM module which upcoming nodes on the topological path is needed, expressed as a list of node IDs from the latest node reached on the path to the farthest node inside the radius R (to limit the size of the list). The required nodes are identified by the graph-based SLAM-MM module with Heuristic 2 either to remain in WM or to be retrieved from LTM to extend the local map. The maximum number of retrieved nodes per map update is limited to M because this operation is time consuming as it needs to load nodes from LTM. M is set based on the hardware on which LTM is saved and according to the maximum velocity of the robot: for instance, if the robot is moving at the same speed or less as when it traversed the same area the first time, $M = 1$ would

suffice to retrieve nodes on the path without having to slow down to wait for nodes not yet retrieved.

Extending the local map with nodes of the topological path is important for the robot to localize itself using the Appearance-based Loop Closure Detection module or using the Proximity Detection module, making it able to follow the topological path appropriately. As the robot moves and new local maps are created, TPP always looks for the farthest node of the topological path that can be reached in the local map to update the current pose sent to MPP module. If new nodes are retrieved from LTM on the topological path, then the farthest pose is sent to MPP. TPP also detects changes in the local map after graph optimization (e.g., when new loop closures are detected): if so, the updated position of the current pose is sent to MPP.

Up to a ratio O of the WM size, nodes identified by the planner and located in the radius R from the robot's current position are immunized to be transferred, with R being the sensor range.

Figure 3.7 presents an example of the interaction between MPP and TPP to reach a goal G. While the robot is moving, TPP always sends the farthest pose P of the node on the topological path (purple links) in the local map. An occupancy grid is assembled with the laser scans contained in the nodes of the local map. MPP uses this occupancy grid to plan a trajectory (yellow arrow) to P. To keep the WM size constant, as nodes are retrieved from LTM on the path, older nodes are transferred to LTM. To follow the path appropriately, proximity links are detected to correct the map as the robot moves, otherwise the situation explained by Fig. 3.4a would happen.

TPP iterates by sending poses until the node of the goal (under a goal radius D expressed in m) is reached. Finally, handling situations where the environment has changed too much for proper localization must be taken into consideration. If no loop closures and proximity detections occur when following a path, a temporary link is added between the current node and the closest one in the path so that the topological path is always linked to the current node in the local map. Without this link, if previous nodes between the current node and those of the topological path are transferred to LTM, the local map would be divided and the nodes of the path would not be in the local map anymore. This temporary link is removed when a new link is added between the current node and the closest one in the path or when the goal is reached. If the robot has not reached the current pose set to MPP after F iterations of SPLAM-MM (e.g., MPP cannot plan to the requested pose because of the presence of a new obstacle or because the robot cannot

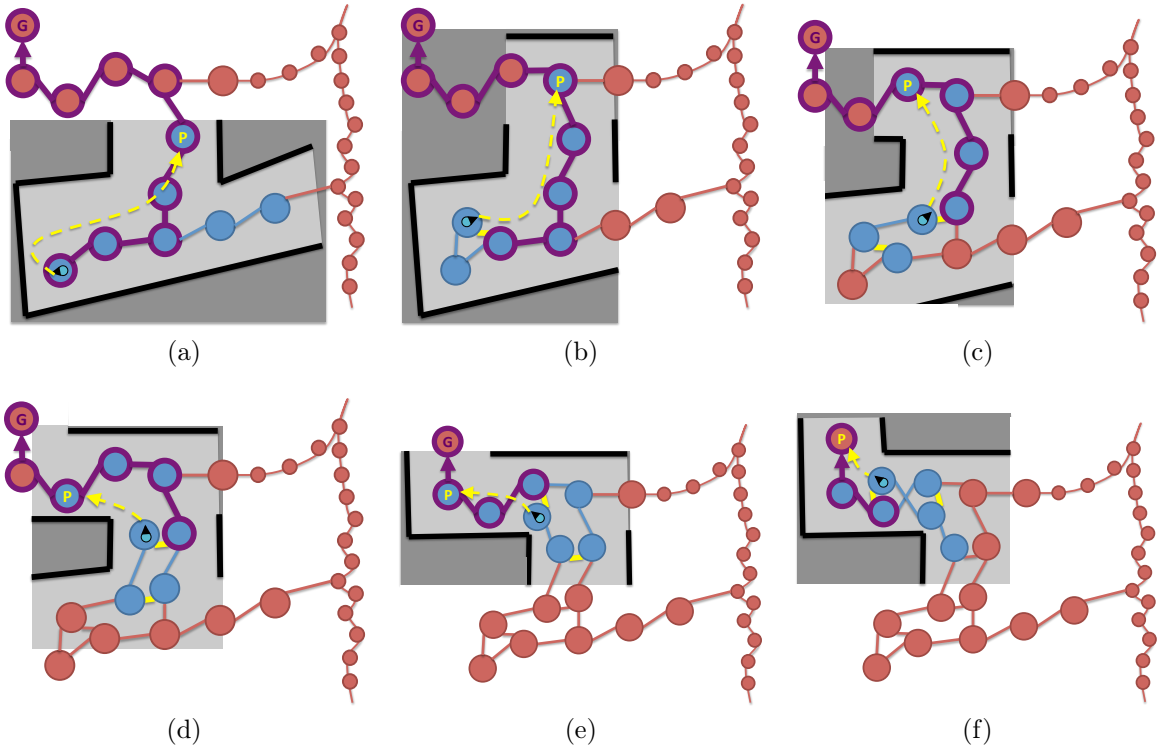


Figure 3.7 Interaction between TPP and MPP for path planning. The goal is identified by the purple G. The topological path is shown with purple links. The dashed yellow arrow is the trajectory computed by MPP to the targeted poses designated by the yellow P. Light gray, dark gray and black areas of the occupancy grid represent free, unknown and occupied cells, respectively. Blue nodes are in WM, and red nodes are in LTM. Yellow links are proximity links.

localize itself on the path), TPP chooses another pose on the upcoming nodes and sends it to MPP. If all the upcoming nodes cannot be reached, TPP fails and sends a status message to its connected modules so that they can be notified that the goal cannot be reached.

3.3.6 Patrol Module

We implemented the Patrol module to generate navigation goals, referred to as waypoints so that the robot is programmed to continuously patrol an area. The Patrol module receives waypoints as inputs and sends them successively to TPP. By examining TPP's status messages, Patrol can know when a goal is reached or if TPP has failed. Whenever the status indicates that the goal is reached or not, the Patrol module sends the next waypoint, and restart to the first one once the whole list has been processed.

3.4 Results

Table 3.1 shows the parameters used for the trials [1]. The acquisition time A used is 1 sec (i.e., the map update rate is 1 Hz), which set the maximum online time allowed to process each node added to the map. For the trials, T is set to 200 ms to limit CPU usage for SPLAM-MM to around 20%, to make sure that higher frequency modules (acquisition of Sensor Data acquisition and MPP) can run at their fixed frequency of 10 Hz. The robot is relatively moving at the same velocity during the trials, and therefore M is fixed to 2 to make sure that nodes on a planned path are retrieved fast enough to avoid having the robot wait for nodes still in LTM. All computations are done onboard on the robot, which is equipped with a 2.66 GHz Intel Core i7-620M and a 128 GB SSD hard drive (on which the LTM is saved).

To define the area over which the robot had to patrol, during session 1 we first teleoperated the robot and defined four waypoints (WP1 to WP4). There were no people in the environment during the teleoperation phase. After reaching WP4, the autonomous navigation phase is initiated by sending the waypoints to the Patrol module. Figure 3.8 illustrates the four waypoints on the global map and the first planned trajectory by TPP (purple path) from the current position of the robot (WP4) to WP1. To come back to WP1, the robot had to follow the path in the opposite direction from when these nodes were created. Proximity detection made it able to follow the path appropriately. To see more clearly the effect of proximity links, Figure 3.9 shows the maps after reaching WP1 with and without graph optimization. Navigation would not have been possible without proximity links: the local map would have look like the map in (b) without the yellow links because no appearance-based similarities would have been found with nodes from the map on the planned path. When reaching WP1, the Patrol module sends the next waypoint (WP2), making the robot continue patrolling.

Every 45 minutes or so of operation, the robot was manually shutdown and moved to the battery charger near WP1. Once recharged, a new session of SPLAM-MM was initiated, creating a new node in STM with odometry reset, while preserving the nodes in WM and LTM. As the robot was initialized in the area of WP1 for each session, loop closures were found, connecting and optimizing the new map with nodes created from previous sessions, and allowing the Patrol module to provide waypoints as navigation goals to patrol the area. Overall, 11 indoor mapping sessions were conducted, for a total distance of 10.5 km lasting 7.5 hours of operation spent over two weeks. The robot did 111 patrolling cycles (i.e., traversing from WP1 through WP2, WP3, WP4 and coming back to WP1).

1. In comparison with [Labbé et Michaud, 2013], $T = T_{time}$, $S = T_{STM}$ and $Y = T_{similarity}$.

Table 3.1 Parameters used for the trials

Acquisition time	A	1 sec
ICP correspondence ratio	C	0.3
Radius of the goal area	D	0.5 m
TPP iterations before failure	F	10
Loop closure hypothesis threshold	H	0.11
Minimum RANSAC visual word inliers	I	5
Close nodes radius	L	0.5 m
Maximum retrieved close nodes	M	2
Heuristics 2 close-by nodes ratio	O	0.25
Laser scan range	R	4 m
STM size	S	20
Time limit	T	200 ms
Maximum obstacle height	U	0.4 m
Similarity threshold	Y	0.3
Ground segmentation maximum angle	Z	0.1 rad

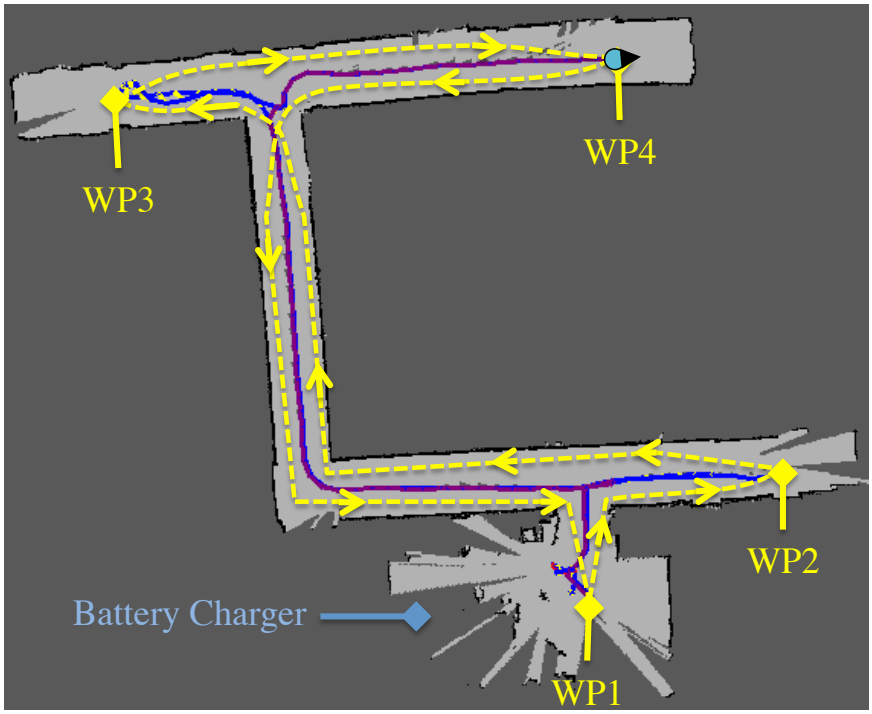
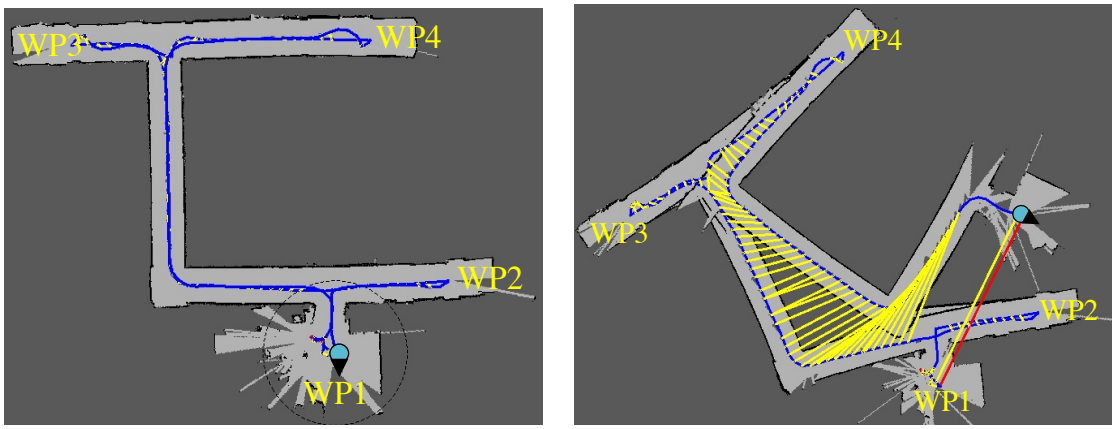


Figure 3.8 Waypoints WP1 to WP4 identified on the global map. The purple path is the first path planned by TPP from the WP4 to WP1.



(a) (b)

Figure 3.9 Global maps, optimized and not optimized, after reaching WP1. Yellow and red links are proximity and loop closure links, respectively.

The sessions were conducted during office hours, with people walking by. A total of 139 people were encountered by the robot while patrolling. Figure 3.10 illustrates the dynamic conditions and some of the obstacles that the robot had to deal with during the trials.

The main goal of the trials is to see how SPLAM is influenced by memory management over long-term operation, only having the local map for online processing. This can be illustrated by looking at the influences of memory management on SPLAM, interactions between TPP and MPP, and the influences of LTM on TPP. As the robot is continuously adding new nodes, the trials also demonstrate how SPLAM-MM works in an unbounded environment.

3.4.1 Influences of MM on SPLAM

Figure 3.11 shows a typical navigation result when reaching the time limit T, thus limiting the size of the local map used for online navigation. This example shows the path planned between WP4 and WP1 after 4.7 hours of operation. The local maps used for online planning, localization and mapping are shown for different time steps along the trajectory. At $t = 17031$ sec, the planned path had 67 nodes and was 33 m long. It took 1.3 sec to be generated by TPP and to have the first pose on the path sent to MPP. The laser scan range R is delimiting the upcoming nodes on the path provided by TPP. As the robot navigates in the environment, the farthest available pose in the local map on the path (end of the cyan line) is sent from TPP to MPP. Upcoming nodes, if they are not in WM, are retrieved to make the robot able to localize itself (though loop closures and proximity detections) on the path. Looking at how the local map changes in these snapshots, notice

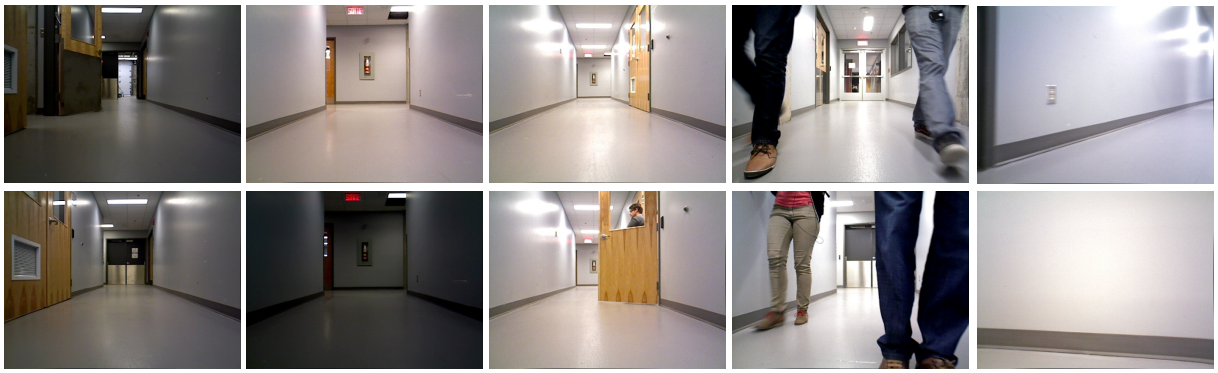


Figure 3.10 Events that occurred during the trials: a) open and closed doors between traversals; b) camera exposure that led to the extraction of different visual features, making it difficult to find loop closures; c) someone opening a door while the robot is navigating; d) people walking around or blocking the robot; e) featureless images on which loop closure detection cannot work.

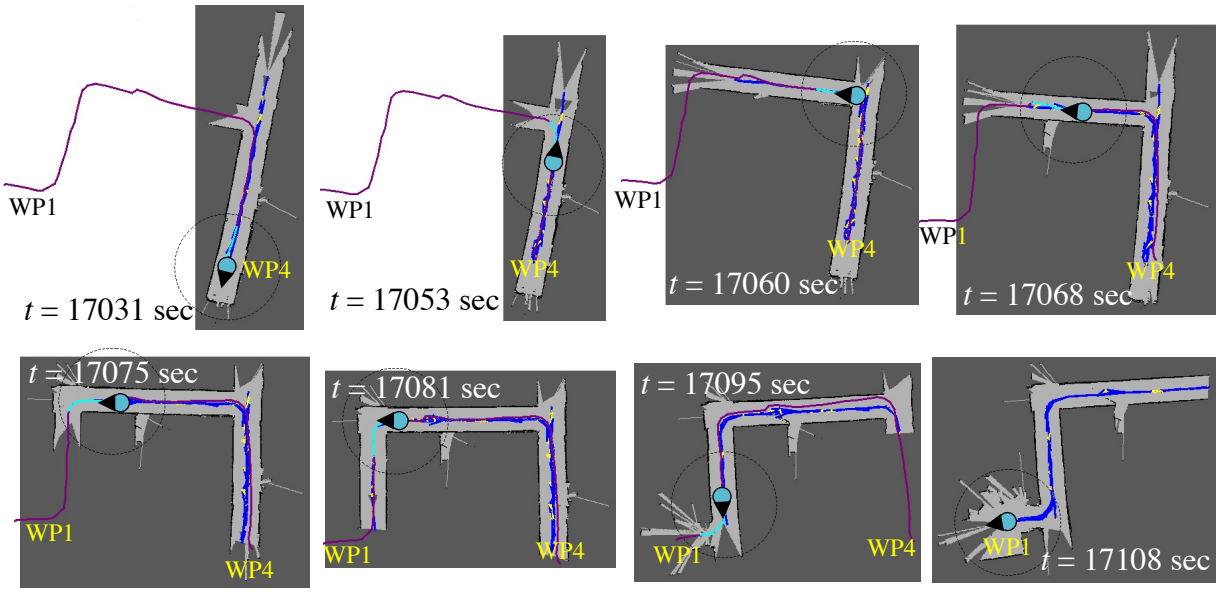


Figure 3.11 Example of the effect of memory management when travelling from WP4 to WP1 after 4.7 hours of operation. The path planned is shown in purple. The small colored icon represents the robot position at each time step. The dotted circle around the robot position illustrates the laser scan range R. The cyan lines represent the upcoming nodes on the planned path.

how starting from $t = 17075$ sec, the initial portion of the path is transferred in LTM to keep the size of the WM relatively constant. At $t = 17108$ sec, the robot reached WP1.

Figure 3.12 compares the images between each waypoint and the final position of the robot at the waypoints. The robot successfully reached the waypoints (within D as the goal radius) 445 out of 446 times. For WP2, WP3 and WP4, the robot always came

from behind the waypoint, and as soon the robot reached the waypoint within a D radius, TPP detected that the goal was reached. This explains why all the poses are behind the waypoints but inside the goal radius D. Similarly, for WP1, the robot came from behind from a slightly different direction. Spurious poses on the right part of the circle are those where there was an obstacle that caused the robot to avoid it, making it reach the waypoint from a different direction. The one time the robot failed to reach a waypoint is because someone blocked the robot for a long time, making TPP failed after F attempts of reaching the upcoming nodes: a failure status message was then sent to the Patrol module to provide the next waypoint. The person left soon after the next waypoint was sent, and the robot reached the new waypoint provided.

Figure 3.13 illustrates the evolution of the number of nodes in WM and online processing time over the 11 mapping sessions. Processing time includes all SPLAM-MM modules except MPP which was running concurrently on a separate process (its processing time is only dependent of the local map size). As explained in Section 3.3.5, TPP occurs offline and only when a new goal is received from the Patrol module, and is examined in Section 3.4.3. Figure 3.13a illustrates that the number of nodes in WM and the local map was identical until T sec was reached. After that, nodes were transferred to LTM to limit the WM size for online processing, which is satisfied as shown by Figure 3.13b. Processing time also remained well under the acquisition time A.

3.4.2 TPP-MPP Interactions

To illustrate with a concrete example of the situation described in Figure 3.7, Figure 3.14 presents an example of consecutive poses sent by TPP to MPP while nodes from LTM are retrieved for the planned path. The red arrow shows the pose of the farthest node on the path (the direction of the arrow shows the orientation of the pose). The red line represents the trajectory computed by MPP from the current position of the robot to its targeted pose, combined with obstacle avoidance. The blue lines represent the local map. In Figure 3.14a, the targeted pose is on a node traversed backward (as shown by the arrow pointing backward). Between a) and b), the local map was updated with nodes loaded from LTM of the topological path. The targeted pose was updated farther on the path and at the same time, the occupancy grid was extended to previously mapped areas and MPP recomputed its trajectory. The robot could then move farther toward its goal and the nodes retrieved were used for proximity detection to correctly follow the planned path.

To also illustrate the importance of obstacle detection described in Figure 3.6, Figure 3.15 presents an example where an unexpected obstacle was encountered: as the laser

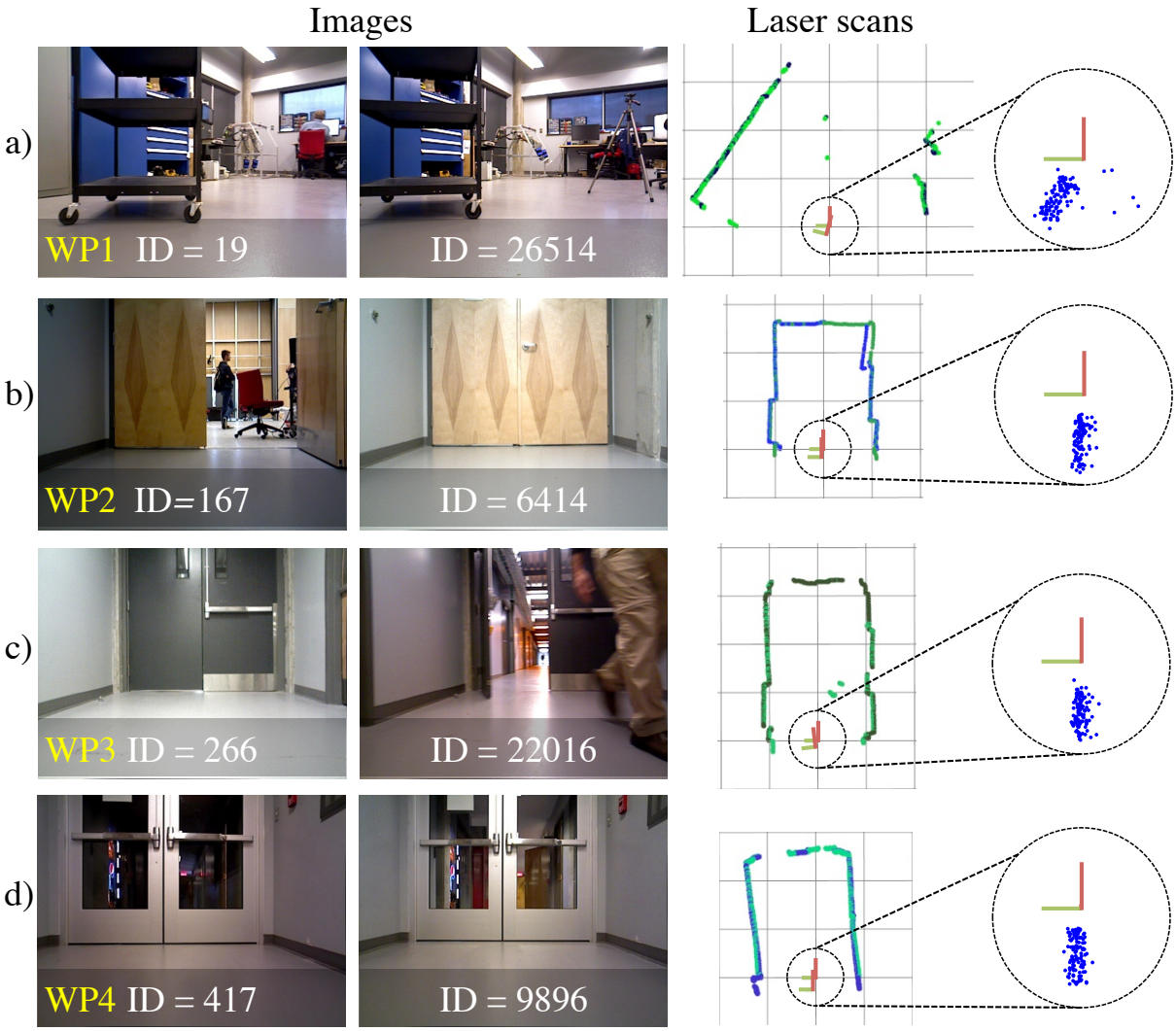
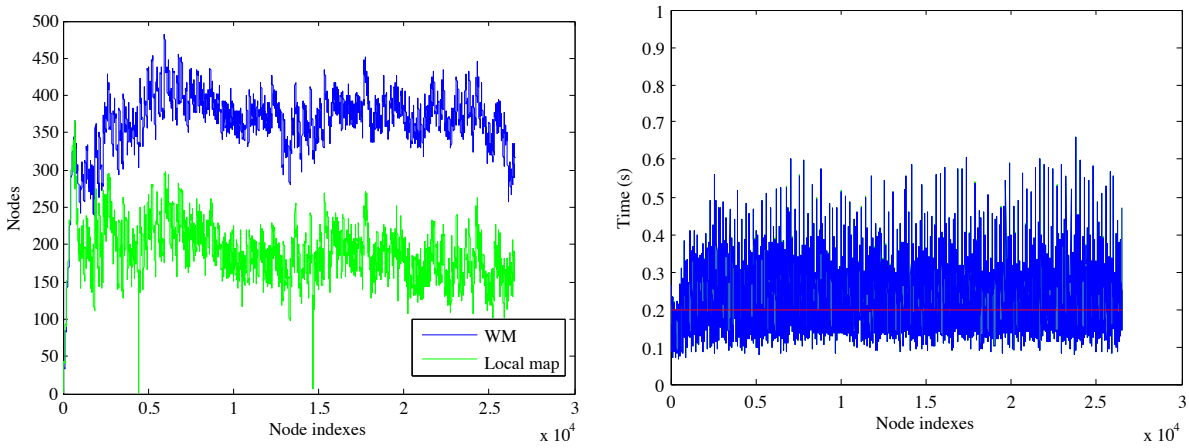


Figure 3.12 Comparison of the corresponding images between the waypoint (left image) and at the last pose reached on one of the planned path (right image) for the waypoints. The top view grid shows the laser scan readings and referentials of the waypoint's nodes (at the origin of the grid) and the final node. The zoomed portions represent the final poses of the robot (represented by blue dots), for all paths planned for each waypoint. The circle represents the goal radius D, and the grid's cells used for visualization have a width of 1 m.

rangefinder is 0.4 m above the ground, the forklift could only be detected using the RGB-D camera. MPP planned a slightly different path (orange) that the one planned by TPP (pink) to avoid the obstacle.

3.4.3 Influences of LTM on TPP

Although Figure 3.13 demonstrates that SPLAM-MM is able to satisfy online constraints on a map increasing linearly in size (i.e., not bounded to a maximum size of environment),



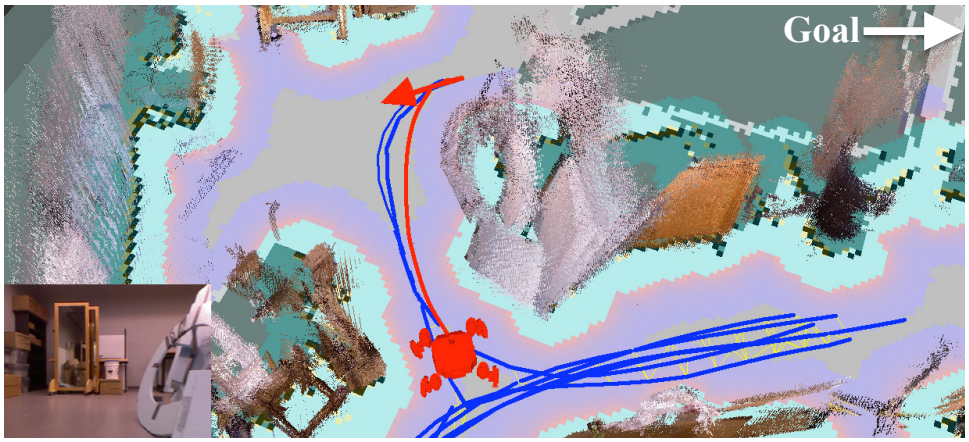
(a) Number of nodes in WM and in the local map.

(b) Processing time (the horizontal line represents $T = 0.2$ sec).

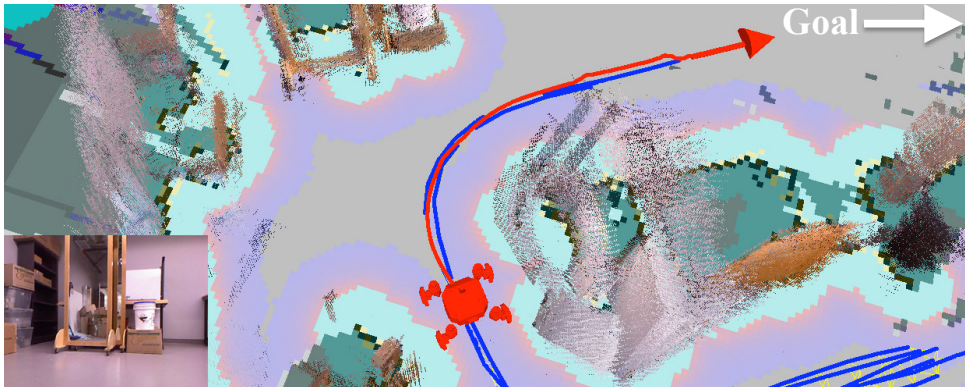
Figure 3.13 Memory size and total processing time over the 11 mapping sessions.

memory used by LTM and consequently TPP planning time increase linearly. For example, at the end of experiment, LTM contains 24002 nodes and 113368 links. All raw sensor data in the nodes were also saved in the LTM's database (for debugging and visualization purposes), including RGB image (JPEG format) and depth image (PNG format) of each node. The final database took 6.7 GB of hard drive space. With as many links at the end of the experiment, TPP required 2.4 sec to compute a plan to the next waypoint. In term of memory usage and planning time, LTM must be somewhat limited over time when revisiting the same areas.

As a solution to limit LTM memory growth, nodes from STM can be merged when moved to WM if they have loop closure and/or visual proximity links. We studied this possibility by adding a graph reduction algorithm to STM, to remove the node from the graph and to add its neighbor links to the corresponding old node(s). Algorithm 1 summarizes the approach used to maintain the graph at the same size (same number of removed links and nodes than added) if there are many successive nodes with loop closure or visual proximity links. If two nodes of a same location do not have similar images (i.e., they don't have loop closure or visual proximity links), they will not be merged, thus still keeping a variety of different images representing the same location. To make sure nodes to be merged are still in WM (to avoid to modify the LTM), nodes having a link to a node in STM are identified as nodes that must stay in WM (similarly to Heuristic 2). Figure 3.16 shows how links are merged between the node moved to WM and its corresponding node(s) linked by loop closure link. In a), the purple node has two loop closure links. On graph reduction, its



(a)



(b)

Figure 3.14 Example of poses sent by TPP to MPP while nodes from LTM are retrieved for the planned path. The goal of the path is somewhere outside these images in the direction shown by Goal. The bottom left images shows the actual RGB image from the RGB-D camera. The blue lines are nodes and links of the local map. The red line is the computed trajectory from MPP using the local map's occupancy grid from its current pose (red arrow). The RGB point cloud and the occupancy grid are created using RGB-D images and laser scans stored in nodes from the local map, respectively. In a), the robot is following the red trajectory. In b), some nodes are retrieved from LTM and a new trajectory is computed to move further on the path toward the goal.

two neighbor links (blue) are merged with the loop closure links (red) by multiplying the corresponding transformations together, creating merged neighbor links (orange). In this case, the same number of links are added than those removed but one node is removed. In b), the green node has only one neighbor link (with the cyan node), then the loop closure link is only merged with it, creating only one link and four are removed. Merged neighbor links are ignored to be merged again to limit the number of links. In c), the cyan node does not have any loop closure and no graph reduction is done.

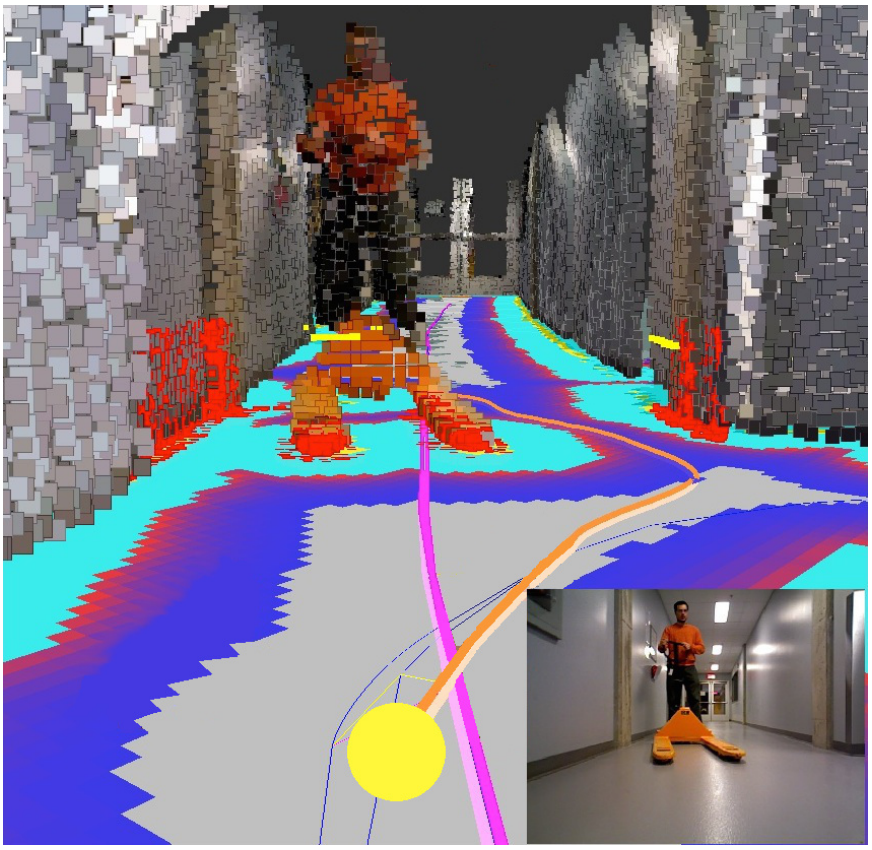


Figure 3.15 Example where MPP plans a slightly different path (orange) than the one provided by TPP (pink). The yellow dot is the current position of the robot and the lower right image is the corresponding RGB image.

To test this idea, data from the 11 sessions were processed again to test the influences of the graph reduction approach using real data acquired by the robot. Note that even though graph reduction was validated offline, we carefully monitored the experiment manually to make sure that the robot could still localize itself correctly on the planned paths.

Figure 3.17 shows a comparison of the final global map without and with graph reduction. The zones with less blue links indicate that there were many nodes merged. The zones with more blue links are where nodes were not merged, because of a lack of features or because of obstacles: the robot was not able to localize itself perfectly on the paths every time, thus adding new nodes to the map.

Figure 3.18 illustrates TPP planning time corresponding to LTM size with and without graph reduction. As the LTM became larger, TPP planning time increased: with graph reduction, TPP planning time was reduced by 89% for the last path planned (272 ms instead of 2.4 sec). Figure 3.19 illustrates hard drive usage with and without graph reduction. Extrapolating linearly memory usage with a 100 Gb hard drive, the robot

Algorithm 1 Graph Reduction

$o \leftarrow$ node moved to WM
$\boldsymbol{m} \leftarrow$ loop closure and visual proximity links of o
if $\boldsymbol{m}$ is not empty **then**
  $\boldsymbol{n} \leftarrow$ neighbor links of o
  for all m in $\boldsymbol{m}$ **do**
    $o_m \leftarrow$ node pointed by m
    for all n in $\boldsymbol{n}$ **do**
      $o_n \leftarrow$ node pointed by n
      $t \leftarrow m^{-1} \cdot n$
      Add t to o_m
      Add t^{-1} to o_n
    end for
  end for
  Remove o from the graph
end if

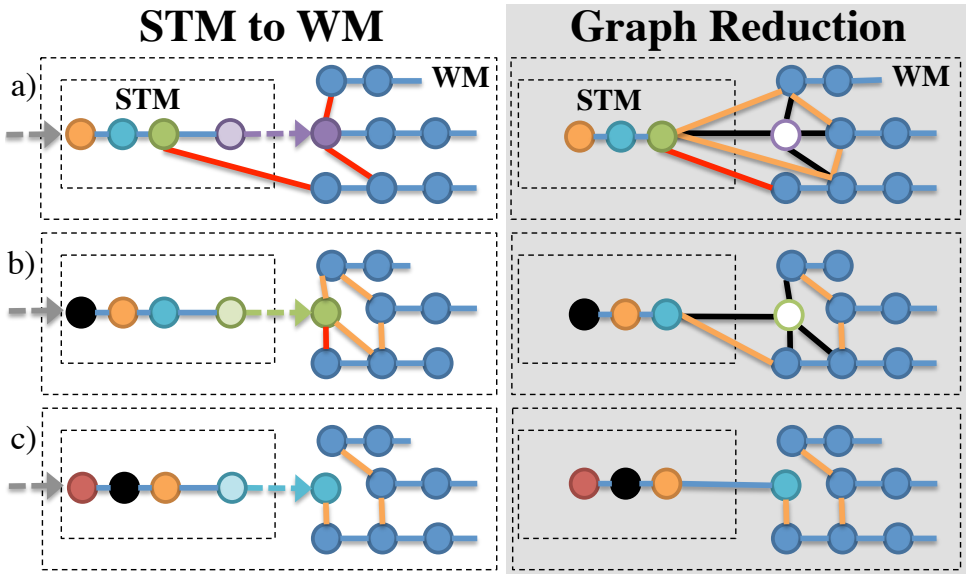


Figure 3.16 Three examples illustrating how the graph reduction algorithm works. Blue, red and orange links represent neighbor, loop closure and merged neighbor links, respectively. Black links and white nodes are those removed using graph reduction. The left column shows the rightmost node (the oldest) of STM moved to WM. Then on the right column, this node is removed if it has a loop closure link.

could navigate online approximately 110 hours without graph reduction before filling up the hard drive. When debugging data (not used for navigation) are not recorded in the database, this estimate would increase to approximately 33 days (800 hours). This means that if the robot is always visiting new locations at a mean velocity of 1.4 km/h (as in this experiment), it could travel up to 1120 km to map environments online. When graph

reduction is used, debugging data are not saved and having the robot always revisiting the same areas like in this experiment, it could do SPLAM continuously for about 130 days before reaching the hard drive capacity.

3.5 Discussion

In terms of processing time, results show that SPLAM-MM is able to satisfy online processing requirements independently of the size of the environment, by transferring in LTM portions of the map which then cannot be used for loop closure detection, proximity detection and graph optimization. Results show also that path following is still possible in such conditions by incrementally retrieving locations on the planned path. Thus, as shown in Section 3.4.3, the current hardware limitation of the system for long-term continuous SPLAM is hard drive capacity, not computation power.

To successfully follow a path, results demonstrate the importance of adding loop closure and/or proximity links with nodes on the planned path to localize the robot in the map. In our trials, the robot navigated indoor where static structures (e.g., walls) were most of the time visible using the laser rangefinder. However, in large empty spaces where the laser rangefinder would not be able to perceive nearby structures, it would be difficult for the robot to follow a path if appearance-based loop closure detection and visual proximity detection do not occur. A laser rangefinder with larger perceptual range or a 3D LIDAR sensor like the Velodyne could be used to increase perceptual range. For a lower cost solution, using a camera facing backward could be useful to allow the robot to detect similarities in images when traversing a path in opposite direction [Carrera *et coll.*, 2011]. Without adding new sensors, TPP could also stop sending new poses when no loop closure links or proximity links occur for a while. If no loop closures were found over the next few meters, it would be possible to wait for the robot to rotate at this location so that it can look backward, increasing its chance to detect a loop closure to correct its position on the planned path and then generate a new pose. A similar recovery approach is presented in [Milford et Wyeth, 2010], where an exploration phase is triggered to re-localize the robot when failing to follow the planned path. Also, to be more robust to dynamic environments where there are cyclic changes over time, TPP could select nodes that match better the current time of the day rather than the most recent ones, to increase localization success as in [Krajník *et coll.*, 2016].

In comparison with large empty environments, those in which a lot of dynamic changes occur (e.g., navigating through a crowd) would also make simultaneous planning and localization more difficult. For instance, mapping the area in session 1 without people

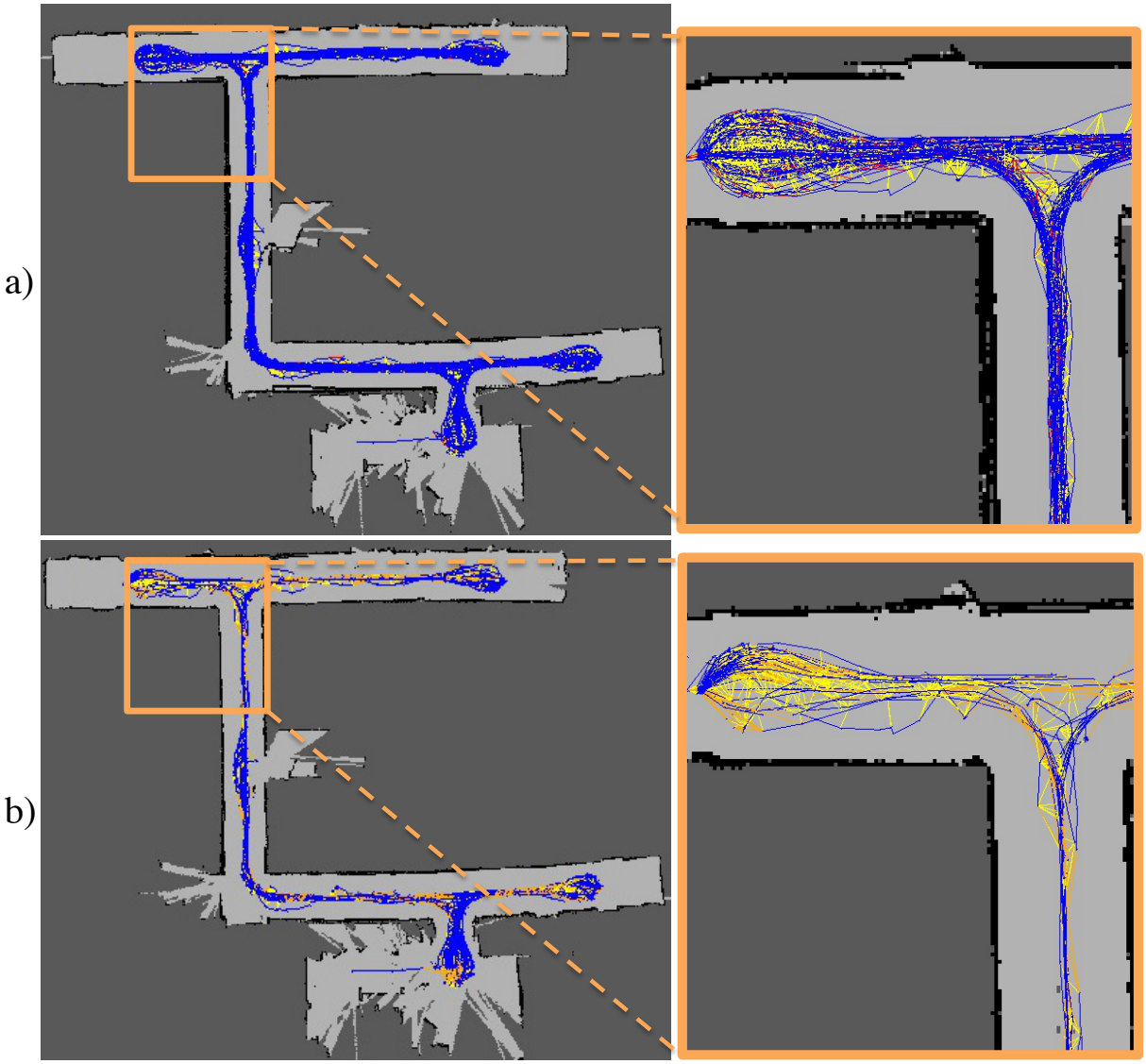


Figure 3.17 Comparison between the global maps a) without graph reduction (24002 nodes and 113368 links); b) with graph reduction (6059 nodes and 18255 links).

walking by helped the robot acquire the static structures of the environment since they were not hidden by people. These static structures facilitate localization when the robot comes back to these areas later one. If these static structures were previously occluded, they would be added to the map as the robot comes back to these areas (obviously if people are no longer in the robot's field of view). If people partially occlude the robot's sensors over a long distance, localization would still be possible but would occur less frequently.

For online multi-session mapping with our memory management approach, the worst case is when all nodes of a previous map are transferred to LTM before a loop closure is

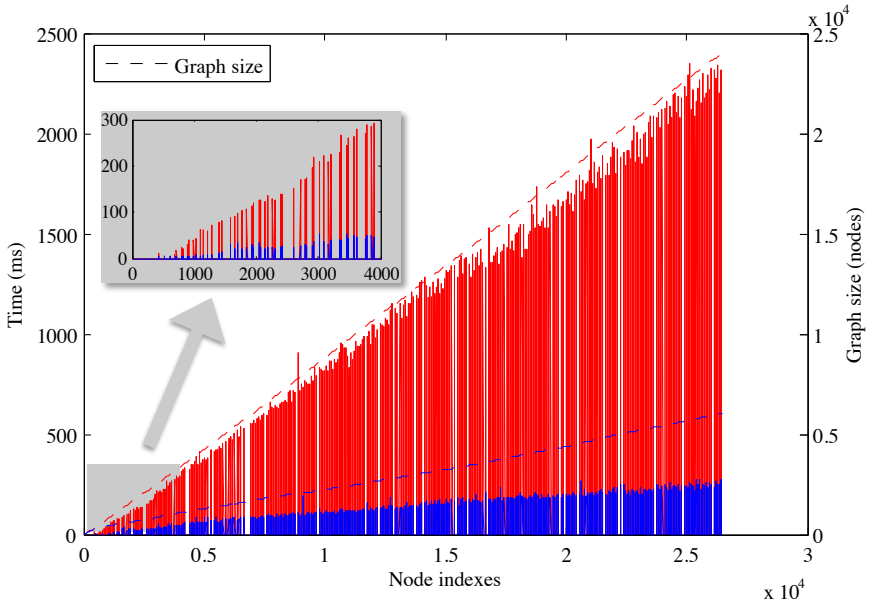


Figure 3.18 Comparison of TPP planning time and LTM size, with (blue) and without (red) graph reduction. The peaks in the zoomed section show more precisely when a planning is done (when a waypoint is reached).

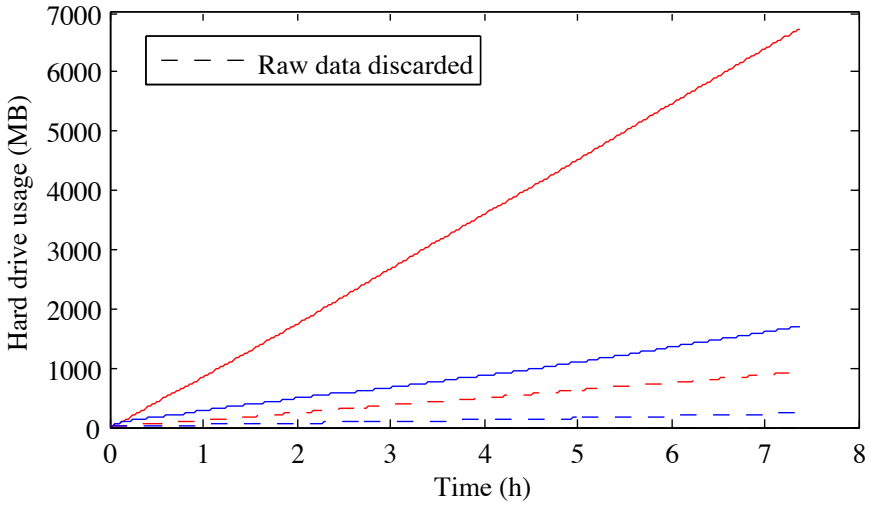


Figure 3.19 Comparison of hard drive usage with (blue) and without (red) graph reduction. The dashed curves represents results without saving in database the debugging data (i.e., raw RGB and depth images).

detected [Labbé et Michaud, 2013]. This results in definitely ignoring the previous map and disabling at the same time the ability to plan paths to a location in it. To avoid this problem, an additional heuristic could be to keep in WM at least one discriminative node for each map. However, if the number of mapping sessions becomes very high (e.g., thousands of sessions), these nodes would definitely have to be transferred in LTM to satisfy online processing requirements. A strategy that makes the robot explore potential

paths to link maps together would then be useful, and maps that could not be linked would eventually be unretrievable.

In the trials conducted, no invalid loop closures were detected, avoiding to corrupt the map with erroneous loop closure links. If this happens, graph optimization approaches such as [Latif *et coll.*, 2013; Lee *et coll.*, 2013; Sunderhauf et Protzel, 2012] deal with possible invalid matches, and could be used to increase robustness of SPLAM-MM. However, these approaches assume that the whole global map is available online, which is not the case here. They could be still used offline at the end of a session.

As shown by Figure 3.15, MPP in SPLAM-MM allows the robot to find an alternative path to reach the targeted pose when possible. However, if the alternative path is outside the local map, re-planning with TPP is required. Some paths may be also blocked temporary or permanently by some dynamic or new static obstacles. An approach similar to [Konolige *et coll.*, 2011] could be used to identify some links as blocked so that TPP cannot plan a path using them. The Patrol module could also manage waypoints that can and cannot be reached.

Finally, the graph reduction approach can reduce significantly the number of nodes and links saved in LTM to reduce TPP planning time. However, because of dynamic events or the lack of features (e.g., Figure 3.10e), new nodes and links will inevitably be added to LTM over time when revisiting the same areas. As an improvement, nodes with featureless image could be merged through a maximum density threshold like in [Milford et Wyeth, 2010], as they cannot be used for loop closure detection. After applying graph reduction on the experimental data, there are still 3068 featureless nodes of 6059 nodes in the global graph, which would reduce by about 50% the remaining graph. However, even by limiting the rate at which the LTM grows, a continuous SLAM approach in unbounded dynamic environments will always add new data over time. A complementary strategy would be to definitely forget some parts of the global map, at the cost of not being able to return to some locations.

3.6 Conclusion

By limiting the nodes of the map available online in WM for loop closure detection, proximity detection and graph optimization, results presented in this paper suggest that the proposed graph-based SPLAM-MM approach is able to meet online processing requirements needed for simultaneous mapping, localizing and planning in multi-session conditions. SPLAM-MM is tightly based on appearance-based loop closure detection, al-

lowing it to naturally deal with the initial state problem of multi-session mapping. To successfully localize on a planned path through areas previously transferred in LTM, memory management allows SPLAM-MM to deal with the necessity of retrieving upcoming nodes on the path in WM. Our code is open source and available at `http://introlab.github.io/rtabmap`.

In future works, more robust failure recovery approaches will be examined to test SPLAM-MM in dynamic environments where the paths could often be blocked (temporally or permanently). We also plan to study the impact of autonomous coverage and exploration strategies, especially how it can actively direct exploration based on nodes available for online mapping. This could be also useful to conduct longer experiments at larger scale.

CHAPITRE 4

RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation

Avant-propos

Auteurs et affiliations :

M. Labbé : étudiant au doctorat, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique.

F. Michaud : professeur, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique.

Date de soumission : 24 juillet 2018

État de l'acceptation : accepté le 19 août 2018 et publié en ligne le 27 octobre 2018 (`https://doi.org/10.1002/rob.21831`)

Revue : *Journal of Field Robotics*

Titre français : RTAB-Map en tant que logiciel libre de SLAM visuel et par télémètre laser pour un fonctionnement 'en ligne' à grande échelle et à long terme

Contribution au document : Cet article présente une description détaillée du logiciel libre RTAB-Map, librairie implémentant l'approche de cartographie, localisation et planification simultanées 'en ligne', à long terme et à grande échelle pour robot mobile, soit la contribution centrale de la thèse. Il présente aussi une comparaison exhaustive, de façons quantitative et qualitative, avec les autres approches de SLAM disponibles dans la communauté robotique. De plus, la librairie est conçue de manière à pouvoir tester la gestion de mémoire sur une multitude de robots ayant des configurations de capteurs différentes. En effet, selon que seulement des caméras sont utilisées ou non et avec ou sans télémètres laser, les performances de navigation peuvent différer. Avec cette version de RTAB-Map, les deux grands paradigmes de SLAM, visuel vs géométrique, peuvent ainsi être comparés avec le même environnement.

Résumé français : Distribué comme logiciel libre depuis 2013, RTAB-Map a commencé comme étant une approche de détection de fermeture de boucle basée sur l'apparence et incluant une gestion de la mémoire pour traiter les opérations 'en ligne', à grande échelle et à long terme. Il a ensuite évolué pour permettre la cartographie et la localisation simultanées (SLAM) sur divers robots et plateformes mobiles. Comme chaque application apporte son lot de contraintes sur les capteurs, les capacités de traitement et la locomotion, un question consiste à savoir quelle approche de SLAM est la plus appropriée en termes de coût, précision, puissance de calcul et facilité d'intégration. Comme la plupart des approches de SLAM sont basées soit sur la vision ou soit sur le télémètre laser, leur comparaison est difficile. Par conséquent, nous avons décidé d'étendre RTAB-Map pour prendre en charge le SLAM visuel et le télémètre laser, fournissant en un seul logiciel un outil permettant aux utilisateurs de mettre en œuvre et de comparer une variété de solutions 3D et 2D pour une large gamme d'applications avec différentes configurations de capteurs sur des robots. Cet article présente cette version étendue de RTAB-Map et son utilisation pour comparer, quantitativement et qualitativement, une large sélection d'ensembles de données populaires (par exemple, KITTI, EuRoC, TUM RGB-D, MIT Stata Center sur le robot PR2), décrivant les points forts et les limites des configurations de SLAM visuels et avec télémètre laser d'un point de vue pratique pour les applications de navigation autonome de robots mobiles.

Abstract

Distributed as an open source library since 2013, RTAB-Map started as an appearance-based loop closure detection approach with memory management to deal with large-scale and long-term online operation. It then grew to implement Simultaneous Localization and Mapping (SLAM) on various robots and mobile platforms. As each application brings its own set of contraints on sensors, processing capabilities and locomotion, it raises the question of which SLAM approach is the most appropriate to use in terms of cost, accuracy, computation power and ease of integration. Since most of SLAM approaches are either visual or lidar-based, comparison is difficult. Therefore, we decided to extend RTAB-Map to support both visual and lidar SLAM, providing in one package a tool allowing users to implement and compare a variety of 3D and 2D solutions for a wide range of applications with different robots and sensors. This paper presents this extended version of RTAB-Map and its use in comparing, both quantitatively and qualitatively, a large selection of popular real-world datasets (e.g., KITTI, EuRoC, TUM RGB-D, MIT Stata Center on PR2 robot), outlining strengths and limitations of visual and lidar SLAM configurations from a practical perspective for autonomous navigation applications.

4.1 Introduction

RTAB-Map, for Real-Time Appearance-Based Mapping [2] [Labbé et Michaud, 2013, 2017], is our open source library implementing loop closure detection with a memory management approach, limiting the size of the map so that loop closure detections are always processed under a fixed time limit, thus satisfying online requirements for long-term and large-scale environment mapping. Initiated in 2009 and released as an open source library in 2013, RTAB-Map has since be extended to a complete graph-based SLAM approach [Stachniss *et coll.*, 2016] to be used in various setups and applications [Chen *et coll.*, 2015; Foresti *et coll.*, 2016; Goebel, 2014; Laniel *et coll.*, 2017]. As a result, RTAB-Map has evolved into a cross-platform standalone C++ library and a ROS package [3], driven by practical requirements such as:

– Online processing: output of the SLAM module should be bounded to a maximum delay after receiving sensor data. For graph-based SLAM in particular, as the map grows, more processing time is required to detect loop closures, to optimize the graph and to assemble the map. Also, integration with other processing modules for control, navigation, obstacle avoidance, user interaction, object recognition, etc. may

2. `http://introlab.github.io/rtabmap`
3. `http://wiki.ros.org/rtabmap_ros`

also limit the CPU time available for SLAM. Having the possibility to limit computation load is therefore beneficial to avoid lagging problems with other modules, and may even be necessary to prevent unsafe situations.

– Robust and low-drift odometry: while loop closure detection can correct most of the odometry drift, in real-world scenarios the robot often cannot properly localize itself on the map, either because it is exploring new areas or that there is a lack of discriminative features in the environment. During that time, odometry drift should be minimized so that accurate autonomous navigation is still possible until localization can occur, to avoid incorrectly overwriting mapped areas (e.g., incorrectly adding obstacles in the entrance of a room, making it a closed area for instance). Estimating odometry with exterioceptive sensors such as cameras and lidars can be very accurate when there are enough features in the environment, but only using one sensing modality can be problematic and prone to localization failures if their tracked features in the environment are no longer visible. Using a mix of proprioceptive (e.g., wheel encoders, inertial measurement units (IMU)) and exterioceptive sensors would increase robustness to odometry estimation.
– Robust localization: the SLAM approach must be able to recognize when it is revisiting past locations (for loop closure detection) to correct the map. Dynamic environments, illumination changes, geometry changes or even repetitive environments can lead to incorrect localization or failure to localize, and therefore the approach should be robust to false positives.
– Practical map generation and exploitation: most popular navigation approaches are based on occupancy grid, and therefore it is beneficial to develop SLAM approaches that can provide 3D or 2D occupancy grid out-of-the-box for easy integration. Also, when the environment is mostly static, it is more practical to do a mapping session and then switch to localization, setting memory usage and saving map management time.
– Multi-session mapping (a.k.a. *kidnapped robot problem* or *initial state problem*): when turned on, a robot does not know its relative position to a previously created map, making it impossible to plan a path to a previously visited location. To avoid having the robot restart the mapping process to zero or localize itself in a previously-built map before initiating mapping, multi-session mapping allows the SLAM approach to initialize a new map with its own referential on startup, and when a previously visited location is encountered, a transformation between the two maps can be computed. This brings the advantages of avoiding remapping the whole

environment when only a small part should be remapped or a new area should be added.

With the diversity of available SLAM approaches, determining which one to use in relation to a specific platform and application is a difficult task, mostly because of the absence of comparative analyses between them. SLAM approaches are generally visual-based [Fuentes-Pacheco *et coll.*, 2015] or lidar-based only [Thrun, 2002], and are benchmarked often on datasets having only a camera or a lidar, but not both, making difficult to have a meaningful comparison between them. It is even more the case when their implementation is either unavailable, only run offline or the required input formats on the robot platform are missing. The Robotic Operating System (ROS) [Quigley *et coll.*, 2009], introduced in 2008, contributes greatly to standardize sensor data format, thus improving interoperability between robot platforms and making it possible to compare SLAM approaches. But still, visual SLAM approaches integrated in ROS are not often tested on autonomous robots: only SLAM by teleoperation or by a human moving the sensor [Dai *et coll.*, 2017; Engel *et coll.*, 2015; Mur-Artal et Tardós, 2017]. This avoids proper *tf* (Transform Library) [Foote, 2013] handling to transform the outputs according to the robot base frame to satisfy ROS coordinate frame convention [4]. It also avoids the need to have map outputs (e.g., 2D or 3D occupancy grid) compatible for the navigation algorithm to plan a path and avoid obstacles. Furthermore, some of the practical requirements outlined above are not always all addressed by the SLAM approaches, thus limiting comparison.

Therefore, since RTAB-Map evolved to handle these practical requirements, we decided to further extend RTAB-Map capabilities to compare visual and lidar SLAM configurations for autonomous robot navigation. RTAB-Map being a loop-closure approach with memory management as its core, it is independent of the odometry approach used, meaning that it can be fed with visual odometry, lidar odometry or even just wheel odometry. This means that RTAB-Map can be used to implement either a visual SLAM approach, a lidar SLAM approach or a mix of both, which makes it possible to compare different sensor configurations on a real robot. This paper describes the extended version of the RTAB-Map library and demonstrates its use to compare state-of-the-art visual and lidar SLAM approaches, and consequently outlining practical limitations between the two paradigms for autonomous navigation.

The paper is organized as follows. Section 4.2 presents a brief overview of popular SLAM approaches currently available, compatible with ROS and that can be used on a robot for comparative evaluations. Section 4.3 presents the main components of the extended ver-

4. `http://www.ros.org/reps/rep-0105.html`

sion of RTAB-Map. Section 4.4 uses RTAB-Map to compare its visual and lidar SLAM configurations in terms of trajectory performance using standard offline and online datasets: the KITTI dataset for outdoor stereo and 3D lidar mapping by autonomous cars; the TUM RGB-D dataset for hand-held RGB-D mapping; the EuRoC dataset for stereo mapping on a drone; and the MIT Stata Center dataset comparing indoor stereo, RGB-D and 2D lidar SLAM configurations on a PR2 robot. Section 4.5 assesses map quality and computation performance variations according to the sensors used, and shows the effect of memory management for online mapping. Finally, Section 4.6 presents, based on the observed results, guidelines derived through the use of RTAB-Map regarding the choice of sensors for autonomous robot SLAM applications.

4.2 Popular SLAM Approaches Available on ROS

There are a great variety of open-source SLAM approaches available through ROS. In this section, we review the most popular ones to outline their characteristics and to situate what RTAB-Map covers in terms of inputs and outputs to handle comparative studies of SLAM approaches.

Let us start with the following lidar approaches:

- GMapping [Grisetti *et coll.*, 2007b] and TinySLAM [Steux et El Hamzaoui, 2010] are two approaches that use a particle filer to estimate the robot trajectory. As long as there are enough estimated particles and the real position error corresponds to the covariance of the input odometry, the particle filter converges to a solution which represents well the environment, particularly for GMapping when there are loop closures. GMapping, being ROS' default SLAM approach, has been widely used to derive a 2D occupancy grid map of the environment from 2D laser scans. Once the map is created, it can be used with Adaptive Monte Carlo Localization [Fox *et coll.*, 1999] for localization and autonomous navigation.
- Hector SLAM [Kohlbrecher *et coll.*, 2011] can create fast 2D occupancy grid maps from a 2D lidar with low computation resources. It has proven to generate very low-drift localization while mapping in real-world autonomous navigation scenarios, like those in RoboCup Rescue Robot League competition [Kohlbrecher *et coll.*, 2016]. It can also use external sensors like an IMU to estimate the robot position in 3D. However, Hector SLAM is not exactly a full SLAM approach as it does not detect loop closures, and thus the map cannot be corrected when visiting back a previous localization. Hector SLAM does not need external odometry, which can be an advantage when the robot does not have one, but can be a disadvantage when

operating in an environment without a lot of geometry constraints, limiting laser scan matching performance.

- ETHZASL-ICP-Mapper [5], based on *libpointmatcher* library [Pomerleau *et coll.*, 2013], can be used to create 2D occupancy grid maps from 2D lidar and an assembled point cloud from 2D or 3D lidars. But similarly to Hector SLAM, the approach does not detect loop closures, thus map errors over time cannot be corrected.
- Karto SLAM [Vincent *et coll.*, 2010], Lago SLAM [6] [Carlone *et coll.*, 2012] and Google Cartographer [Hess *et coll.*, 2016] are lidar graph-based SLAM approaches. They can generate 2D occupancy grid from their graph representation. Google Cartographer can be also used as backpack mapping platform as it supports 3D lidars, thus providing a 3D point cloud output. While mapping, they create sub-maps that are linked by constraints in the graph. When a loop closure is detected, the position of the sub-maps are re-optimized to correct errors introduced by noise of the sensor and scan matching accuracy. Unlike Hector SLAM, external odometry can be provided to get more robust scan matching in environments with low geometry complexity.
- BLAM [7] is a lidar graph-based SLAM that only supports 3D lidar for 3D point cloud generation of the environment. From the online documentation (which is the only documentation available), loop closures seem detected locally by scan matching when the robot visits previous locations, to then optimize the map using GTSAM [Dellaert, 2012]. This means that BLAM is not able to close large loops, for which local scan matching would not be able to appropriately register.
- SegMatch [Dubé *et coll.*, 2016] is a 3D lidar-based loop closure detection approach that can be also used as 3D lidar graph-based SLAM. Loop closures are detected by matching 3D segments (e.g., parts of vehicles, buildings or trees) created from laser point clouds.

In these lidar-based SLAM approaches, only SegMatch can be used for multi-session or multi-robot mapping [Dubé *et coll.*, 2017].

Regarding visual SLAM, many open-source approaches exist but not many can be easily used on a robot (consult [Zollhöfer *et coll.*, 2018] for a review on 3D reconstruction focused approaches). For navigation, to avoid dealing with scale ambiguities, we limit our review to approaches able to estimate the real scale of the environment while mapping (e.g., with stereo and RGB-D cameras or with visual-inertial odometry), thus excluding structure

5. `http://wiki.ros.org/ethzasl_icp_mapper`
6. `https://github.com/rrg-polito/rrg-polito-ros-pkg`
7. `https://github.com/erik-nelson/blam`

from motion or monocular SLAM approaches like PTAM [Klein et Murray, 2007], SVO [Forster *et coll.*, 2014], REMODE [Pizzoli *et coll.*, 2014], DT-SLAM [Herrera *et coll.*, 2014], LSD-SLAM [Engel *et coll.*, 2014] or ORB-SLAM [Mur-Artal *et coll.*, 2015]. The following visual SLAM approaches do not suffer from this scale drift over time.

- maplab [Schneider *et coll.*, 2018] and VINS-Mono [Yi *et coll.*, 2017] have recently been released as visual-inertial graph-based SLAM systems. Using only an IMU and a camera, they can provide visual maps for localization. maplab workflow is done in two steps: the data is recorded during an open loop phase using only visual-inertial odometry; then map management (i.e., loop closure detection, graph optimization, multi-session, dense map reconstruction) is done offline. The resulting visual map can be then used in localization mode afterward. In contrast, VINS-Mono's map management process is done online. For navigation, a local TSDF volume map computed on GPU can be provided for obstacle avoidance and path planning. To keep processing time bounded for large-scale environments, VINS-Mono limits the size of the graph, removing nodes without loop closures first, then removing others depending on the density of the graph.
- ORB-SLAM2 [Mur-Artal et Tardós, 2017] and S-PTAM [Pire *et coll.*, 2017] are currently two of the best state-of-the-art feature-based visual SLAM approaches that can be used with a stereo camera. More recently, ProSLAM [Schlegel *et coll.*, 2017] has been released (only benchmark tools available at this time) to provide a comprehensive open source package using well know visual SLAM techniques. For ORB-SLAM2, it can be also used with a RGB-D camera. They are all graph-based SLAM approaches. For ORB-SLAM2 and S-PTAM, when a loop closure is detected using DBoW2 [Gálvez-López et Tardós, 2012], the map is optimized using bundle adjustment. Graph optimization after loop closure is done in a separate thread to avoid influencing camera tracking frame rate performance. For ProSLAM, loop closures are detected by direct comparison of the descriptors in the map, instead of using a bag-of-words approach. For all these approaches, loop closure detection and graph optimization processing time increases as the map grows, which can make loop closure correction happening with a significant delay after being detected. The approaches maintain a sparse feature map. Without occupancy grid or dense point cloud outputs available out-of-the-box like lidar approaches, they can be then difficult to use on a real robot.
- DVO-SLAM [Kerl *et coll.*, 2013], RGBiD-SLAM [Gutierrez-Gomez *et coll.*, 2016] and MPR [Della Corte *et coll.*, 2017], instead of using local visual features to estimate motion, use photometric and depth errors over all pixels of the RGB-D images. They

can generate dense point clouds of the environment. MPR can be also used with a lidar but it is only an odometry approach. DVO-SLAM lacks of a loop closure detection approach independent of the pose estimate, which makes it less suitable for large-scale mapping.

– ElasticFusion [Whelan *et coll.*, 2016], Kintinuous [Whelan *et coll.*, 2015], BundleFusion [Dai *et coll.*, 2017] and InfiniTAM [Kähler *et coll.*, 2016] are based on truncated signed distance field (TSDF) volume for RGB-D cameras. They can reconstruct online very appealing surfel-based maps, but a powerful computer with a recent Nvidia GPU is required. For ElasticFusion, while being able to process camera frames in real-time for small environments, processing time per frame increases according to the number of surfels in the map. For BundleFusion, global dense optimization time on loop closure detection increases according to the size of the environment. InfiniTAM seems faster to close loops, though processing time for loop closure detection and correction still increases with the size of the environment. While being open-source, these algorithms do not support ROS because they rely on extremely fast and tight coupling between the mapping and tracking on the GPU.

All these previous visual SLAM approaches assume that the camera is never obstructed or that images always have enough visual features to track. Such assumptions cannot be satisfied practically on an autonomous robot where the camera can be fully obstructed from people passing by or when the robot is facing a surface without visual features (e.g., white wall) during navigation. The following visual SLAM approaches are designed to be more robust to these events:

– MCPTAM [Harmat *et coll.*, 2015] uses multiple cameras to increase the field of view of the system. If visual features can be perceived through at least one camera, MCPTAM is able to track the position.
– RGBDSLAMv2 [Endres *et coll.*, 2014] can use external odometry as motion estimation. ROS packages like *robot_ localization* [Moore et Stouch, 2014] can be then used to do sensor fusion (with an extended Kalman filter) of multiple odometry sources for a more robust odometry. RGBDSLAMv2 can generate a 3D occupancy grid (OctoMap[Hornung *et coll.*, 2013]) and a dense point cloud of the environment.

Table 4.1 provides a summary of the open-source ROS-compatible SLAM approaches in relation to their inputs and outputs. The Lidar 3D category includes all point cloud types, including those derived from depth images of a RGB-D camera. Odom refers to odometry input that can be used to help the SLAM approach compute motion estimation. 3D occupancy grid map refers to OctoMap [Hornung *et coll.*, 2013]. Note that ORB-

Table 4.1 Popular ROS-compatible lidar and visual SLAM approaches with their supported inputs and online outputs

	Inputs							Online Outputs			
	Camera				Lidar		Odom	Pose	Occupancy		Point
	Stereo	RGB-D	Multi	IMU	2D	3D			2D	3D	Cloud
GMapping					✓		✓	✓	✓		
TinySLAM					✓		✓	✓	✓		
Hector SLAM					✓			✓	✓		
ETHZASL-ICP					✓	✓	✓	✓	✓		Dense
Karto SLAM					✓		✓	✓	✓		
Lago SLAM					✓		✓	✓	✓		
Cartographer					✓	✓	✓	✓	✓		Dense
BLAM						✓		✓			Dense
SegMatch						✓					Dense
VINS-Mono				✓				✓			
ORB-SLAM2	✓	✓									
S-PTAM	✓							✓			Sparse
DVO-SLAM		✓						✓			
RGBiD-SLAM		✓									
MCPTAM	✓		✓					✓			Sparse
RGBDSLAMv2		✓					✓	✓		✓	Dense
RTAB-Map	✓	✓	✓		✓	✓	✓	✓	✓	✓	Dense

SLAM2 and RGBiD-SLAM do not have any online outputs: they do have a visualizer to see the pose and point cloud, but they do not provide them as ROS topics to other modules out-of-the-box. VINS-Mono does provide current point cloud of odometry but not the map and the TSDF map output is not available through the current project page. The last entry in Table 4.1 situates what inputs can be used and outputs that are provided in the extended version of RTAB-Map presented in this paper. Beside RTAB-Map and RGBDSLAMv2, no visual SLAM approaches provide out-of-the-box occupancy grid outputs required for autonomous navigation. RGBDSLAMv2 [Endres *et coll.*, 2014] is probably the visual SLAM approach sharing the most similarities with RTAB-Map, since both can use external odometry as motion estimation. While they do not combine IMU with camera, they can still use visual-intertial odometry approach with their external odometry input. They can also generate a 3D occupancy grid (OctoMap[Hornung *et coll.*, 2013]) and a dense point cloud for depending modules. However, RTAB-Map can also provide 2D occupancy grid like lidar-based SLAM approaches.

4.3 RTAB-Map Description

RTAB-Map is a graph-based SLAM approach that has been integrated in ROS as the *rtabmap_ros*[8] package since 2013. Figure 4.1 shows its main ROS node called *rtabmap*. The odometry is an external input to RTAB-Map, which means that SLAM can also be done using any kind of odometry to use what is appropriate for a given application and robot. The structure of the map is a graph with nodes and links. After sensor synchronization, the Short-Term Memory (STM) module creates a node memorizing the odometry pose, sensor's raw data and additional information useful for next modules (e.g., visual words for Loop Closure and Proximity Detection, and local occupancy grid for Global Map Assembling). Nodes are created at a fixed rate `Rtabmap/DetectionRate` set in milliseconds according to how much data created from nodes should overlap each other. For example, if the robot is moving fast and sensor range is small, the detection rate should be increased to make sure that data of successive nodes overlap, but setting it too high would unnecessary increase memory usage and computation time. A link contains a rigid transformation between two nodes. There are three kind of links: Neighbor, Loop Closure and Proximity links. Neighbor links are added in the STM between consecutive nodes with odometry transformation. Loop Closure and Proximity links are added through loop closure detection or proximity detection, respectively. All the links are used as constraints for graph optimization. When there is a new loop closure or proximity link added to the graph, graph optimization propagates the computed error to the whole graph, to decrease odometry drift. With the graph optimized, OctoMap, Point Cloud and 2D Occupancy Grid outputs can be assembled and published to external modules. Odometry correction to derive the robot localization in map frame is also available through *tf* [Foote, 2013] `/map`→`/odom`.

RTAB-Map' memory management approach [Labbé et Michaud, 2013] runs on top of graph management modules. It is used to limit the size of the graph so that long-term online SLAM can be achieved in large environments. Without memory management, as the graph grows, processing time for modules like Loop Closure and Proximity Detection, Graph Optimization and Global Map Assembling can eventually exceed real-time constraints, i.e., processing time can become greater than the node acquisition cycle time. Basically, RTAB-Map's memory is divided into a Working Memory (WM) and a Long-Term Memory (LTM). When a node is transferred to LTM, it is not available anymore for modules inside the WM. When RTAB-Map's update time exceeds the fixed time threshold `Rtabmap/TimeThr`, some nodes in WM are transferred to LTM to limit the size of the WM and decrease the

8. `http://wiki.ros.org/rtabmap_ros`

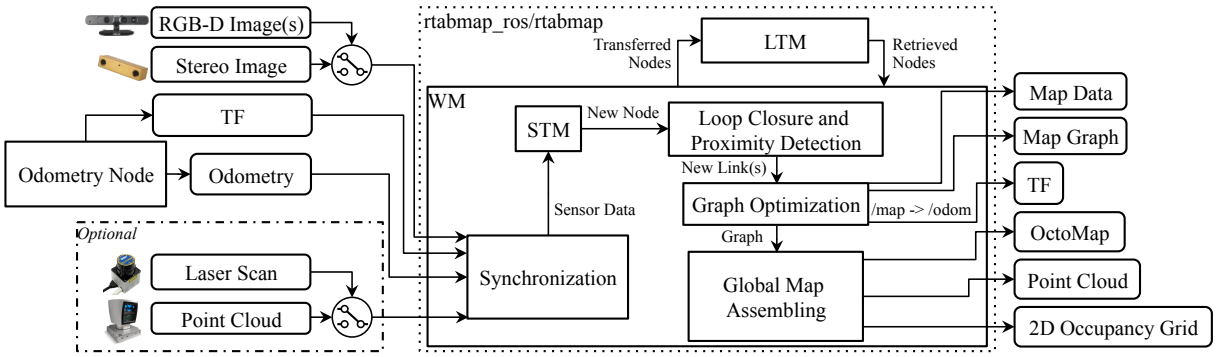


Figure 4.1 Block diagram of *rtabmap* ROS node. The required inputs are: TF to define the position of the sensors in relation to the base of the robot; Odometry from any source (which can be 3DoF or 6DoF); one of the camera inputs (one or multiple RGB-D images, or a stereo image) with corresponding calibration messages. Optional inputs are either a laser scan from a 2D lidar or a point cloud from a 3D lidar. All messages from these inputs are then synchronized and passed to the graph-SLAM algorithm. The outputs are: Map Data containing the latest added node with compressed sensor data and the graph; Map Graph without any data; odometry correction published on TF; an optional OctoMap (3D occupancy grid); an optional dense Point Cloud; an optional 2D Occupancy Grid.

update time. Similarly to the fixed time threshold, there is also a memory threshold `Rtabmap/MemoryThr` that can be used to set the maximum number of nodes that WM can hold. To determine which nodes to transfer to LTM, a weighting mechanism identifies locations that are more important than others, using heuristics such as the longer a location has been observed, the more important it is and therefore should be left in the WM. To do so, when creating a new node, STM initializes the node's weight to 0 and compares it visually (deriving a percentage of corresponding visual words) with the last node in the graph. If they are similar (with the percentage of corresponding visual words over the similarity threshold `Mem/RehearsalSimilarity`), the weight of the new node is increased by one plus the weight of the last node. The weight of the last node is reset to 0, and the last node is discarded if the robot is not moving to avoid increasing uselessly the graph size. When the time or the memory thresholds are reached, the oldest of the smallest weighted nodes are transferred to LTM first. When a loop closure happens with a location in the WM, neighbor nodes of this location can be brought back from LTM to WM for more loop closure and proximity detections. As the robot is moving in a previously visited area, it can then remember the past locations incrementally to extend the current assembled map and localize using past locations [Labbé et Michaud, 2017].

The next sections explain in more details RTAB-Map's pipeline, starting from Odometry Node to Global Map Assembling. Definition of key parameters to configure and use RTAB-Map are provided.

4.3.1 Odometry Node

Odometry Node can implement any kind of odometry approaches from simpler ones derived from wheel encoders and IMU to more complex ones using camera and lidar. Independently of the sensor used, it should provide to RTAB-Map at least the pose of the robot estimated so far in form of an Odometry message [9] with the corresponding *tf*'s transform (e.g., `/odom`→`/base_link`). When proprioceptive odometry is not already available on the robot or when it is not accurate enough, visual or lidar-based odometry must be used. For visual odometry, RTAB-Map implements two standard odometry approaches [Scaramuzza et Fraundorfer, 2011] called Frame-To-Map (F2M) and Frame-To-Frame (F2F). The main difference between these approaches is that F2F registers the new frame against the last keyframe, and F2M registers the new frame against a local map of features created from past keyframes. These two approaches are also implemented for lidars and are referred to as Scan-To-Map (S2M) and Scan-To-Scan (S2S), following the same idea than F2M and F2F but using point clouds instead of 3D visual features. The following sections show how Odometry Node is implemented when one of these visual or lidar odometry approaches is chosen.

Visual Odometry

Figure 4.2 presents RTAB-Map's visual odometry using two colors to differentiate between F2F (green) and F2M (red). It can use RGB-D or stereo cameras as inputs. *tf* is required to know where the camera is placed on the robot so that output odometry can be transformed into the robot base frame (e.g., `/base_link`). If the camera is on the robot's head and the head turns, it does not influence the odometry of the robot base as long as *tf* between the robot's body and the robot's head is also updated. The process works as follows.

– Feature Detection: When a frame is captured, GoodFeaturesToTrack [Shi *et coll.*, 1994] (GFTT) features are detected with a maximum number fixed by `Vis/MaxFeatures` parameter. RTAB-Map supports all feature types available in OpenCV [10], but GFTT has been chosen to ease parameter tuning and get uniformly detected features across different image size and light intensity. For stereo images, stereo correspondences

9. `http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html`
10. `https://opencv.org`

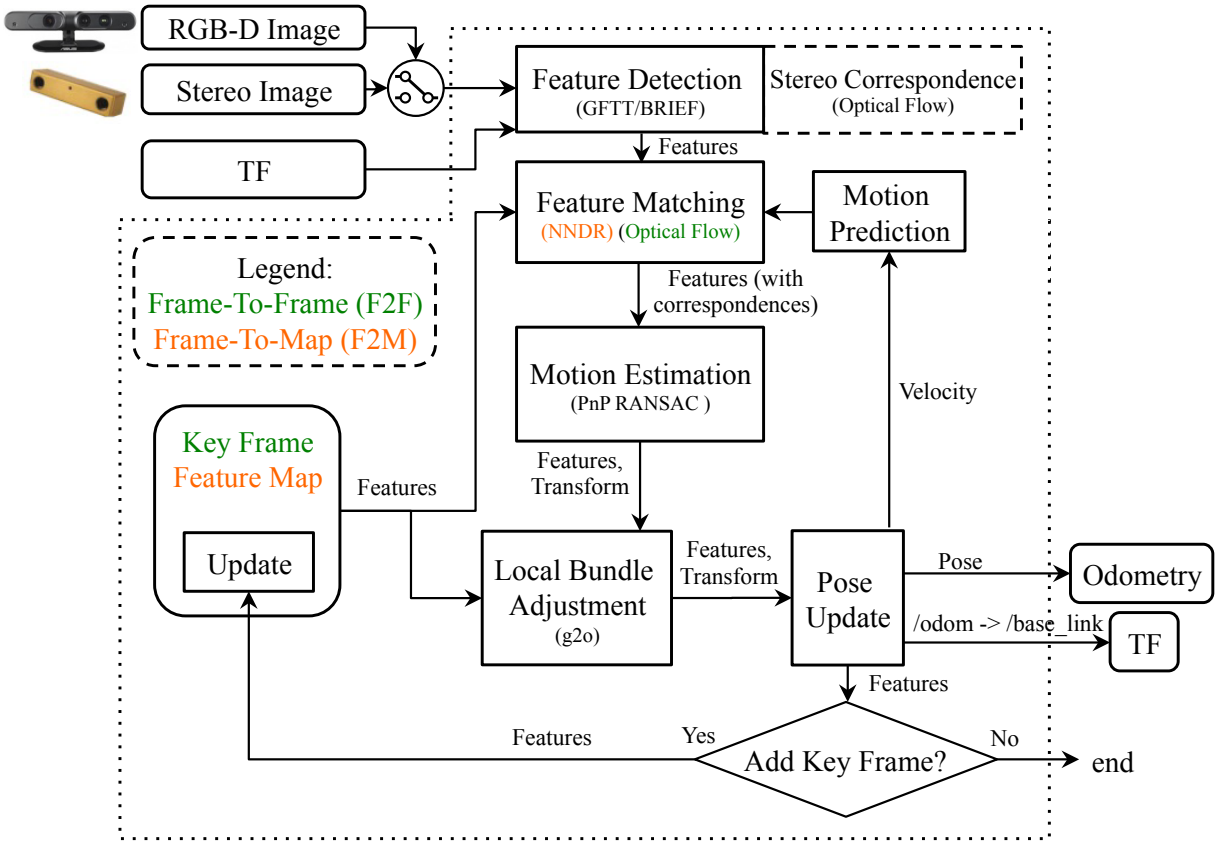


Figure 4.2 Block diagram of *rgbd_ odometry* and *stereo_ odometry* ROS nodes. TF defines the position of the camera in relation to the base of the robot and as output to publish the odometry transform of the base of the robot. The pipeline is the same for a RGB-D camera or a stereo camera, except that stereo correspondences are computed for the later to determine the depth of the detected features. Two odometry approaches can be used: a Frame-To-Frame (F2F) approach in green, and a Frame-To-Map (F2M) approach in orange.

are computed by optical flow using the iterative Lucas–Kanade method [Lucas et Kanade, 1981], to derive disparity per feature between left and right images. For RGB-D images, the depth image is used as a mask for GFTT to avoid extracting features with invalid depth.

– Feature Matching: For F2M, matching is done by nearest neighbor search [Muja et Lowe, 2009] with nearest neighbor distance ratio (NNDR) test [Lowe, 2004], using BRIEF descriptors [Calonder *et coll.*, 2010] of the extracted features against those in the Feature Map. The Feature Map contains 3D features with descriptors from last key frames. NNDR is defined by parameter `Vis/CorNNDR`. For F2F, optical flow is done directly on GFTT features without having to extract descriptors, providing faster feature correspondences against the Key Frame.

- Motion Prediction: A motion model is used to predict where the features of the Key Frame (F2F) or the Feature Map (F2M) should be in the current frame, based on the previous motion transformation. This limits the search window for Feature Matching to provide better matches, particularly in environments with dynamic objects and repetitive textures. The search window radius is defined by parameter `Vis/CorGuessWinSize`, and a constant velocity motion model is used.
- Motion Estimation: When correspondences are computed, the Perspective-n-Point (PnP) RANSAC implementation of OpenCV [Bradski et Kaehler, 2008] is used to compute the transformation of the current frame accordingly to features in Key Frame (F2F) or Feature Map (F2M). A minimum of inliers `Vis/MinInliers` is required to accept the transformation.
- Local Bundle Adjustment: The resulting transformation is refined using local bundle adjustment [Kummerle *et coll.*, 2011] on features of all key frames in the Feature Map (F2M) or only those of the last Key Frame (F2F).
- Pose Update: With the estimated transformation, the output odometry is then updated as well as *tf*'s `/odom`→`/base_link` transform. Covariance is computed using the median absolute deviation (MAD) approach [Rusu et Cousins, 2011] between 3D feature correspondences.
- Key Frame and Feature Map update: If the number of inliers computed during Motion Estimation is below the fixed threshold `Odom/KeyFrameThr`, the Key Frame or Feature Map is updated. For F2F, the Key Frame is simply replaced by the current frame. For F2M, the Feature Map is updated by adding the unmatched features of the new frame and updating the position of matched features that were refined by the Local Bundle Adjustment module. Feature Map has a fixed maximum of features kept temporary (consequently a maximum of Key-Frames). When the size of Feature Map is over the fixed threshold `OdomF2M/MaxSize`, the oldest features not matched with the current frame are removed. If a key frame does not have features in Feature Map anymore, it is discarded.

If for some reasons the current motion of the camera is very different than the predicted one, a valid transformation may not be found (after Motion Estimation or Local Bundle Adjustment boxes), thus features are matched again but without motion prediction. For F2M, features in the current frame are compared to all features in Feature Map, then another transformation is computed. For F2F, to be more robust to invalid correspondences, feature matching with NNDR is done instead of optical flow, and thus BRIEF descriptors have to be extracted. If transformation still cannot be computed, odometry is considered lost and the next frame is compared without motion prediction. The output odometry

pose is set to null with very high variance (i.e., 9999). Modules subscribing to this visual odometry node can then know when odometry cannot be computed.

Note that as odometry in RTAB-Map is independent of the mapping process, other visual odometry approaches have been integrated in RTAB-Map for convenience and ease of comparison between them. The approaches chosen are open-source or provide an application programming interface (API) and can be used as odometry-only. Complete visual SLAM approaches in which it is difficult to split the odometry from the mapping processes cannot be integrated because RTAB-Map take care of the mapping process. Seven approaches have been integrated in RTAB-Map: FOVIS [Huang *et coll.*, 2011], Viso2 [Geiger *et coll.*, 2011], DVO [Kerl *et coll.*, 2013], OKVIS [Leutenegger *et coll.*, 2015], ORB-SLAM2 [Mur-Artal et Tardós, 2017], MSCKF [Sun *et coll.*, 2018] and Google Project Tango. FOVIS, Viso2, DVO, OKVIS and MSCKF are visual or visual-inertial odometry-only approaches, which make them straightforward to integrate by connecting their odometry output to RTAB-Map. ORB-SLAM2 is a full SLAM approach, thus to integrate in RTAB-Map, loop closure detection inside ORB-SLAM2 is disabled. Local bundle adjustment of ORB-SLAM2 is still working, which makes the modified module similar to F2M. The big difference is the kind of features extracted (ORB [Rublee *et coll.*, 2011]) and how they are matched together (direct descriptor comparison instead of NNDR). Similarly to F2M, the size of the feature map is limited so that constant time visual odometry can be achieved (without limiting the feature map size, ORB-SLAM2 computation time increases over time). As ORB-SLAM2 has not been designed (at least in the code available at the time of writing this paper) to remove or forget features in its map, memory is not freed when features are removed, which results in an increasing RAM usage over time (a.k.a. memory leak). To integrate Google Project Tango in RTAB-Map library, *area learning* feature is disabled and its visual inertial odometry is used directly.

Lidar Odometry

Figure 4.3 provides the block diagram of lidar odometry, also using two colors to differentiate between S2S (green) and S2M (red). Using a terminology similar to visual odometry, a key frame refers to a point cloud or a laser scan. The laser scan input is 2D as the point cloud input can be either 2D or 3D. Laser scans can have some motion distortions when the robot is moving during the scan. It is assumed here that such distortions are corrected prior to feed the scan to RTAB-Map. Note that if the laser scanner rotation frequency is high comparatively to robot velocity, laser scans would have very low motion distortion and thus correction can be ignored without significant loss of registration accuracy. The process is described as follows:

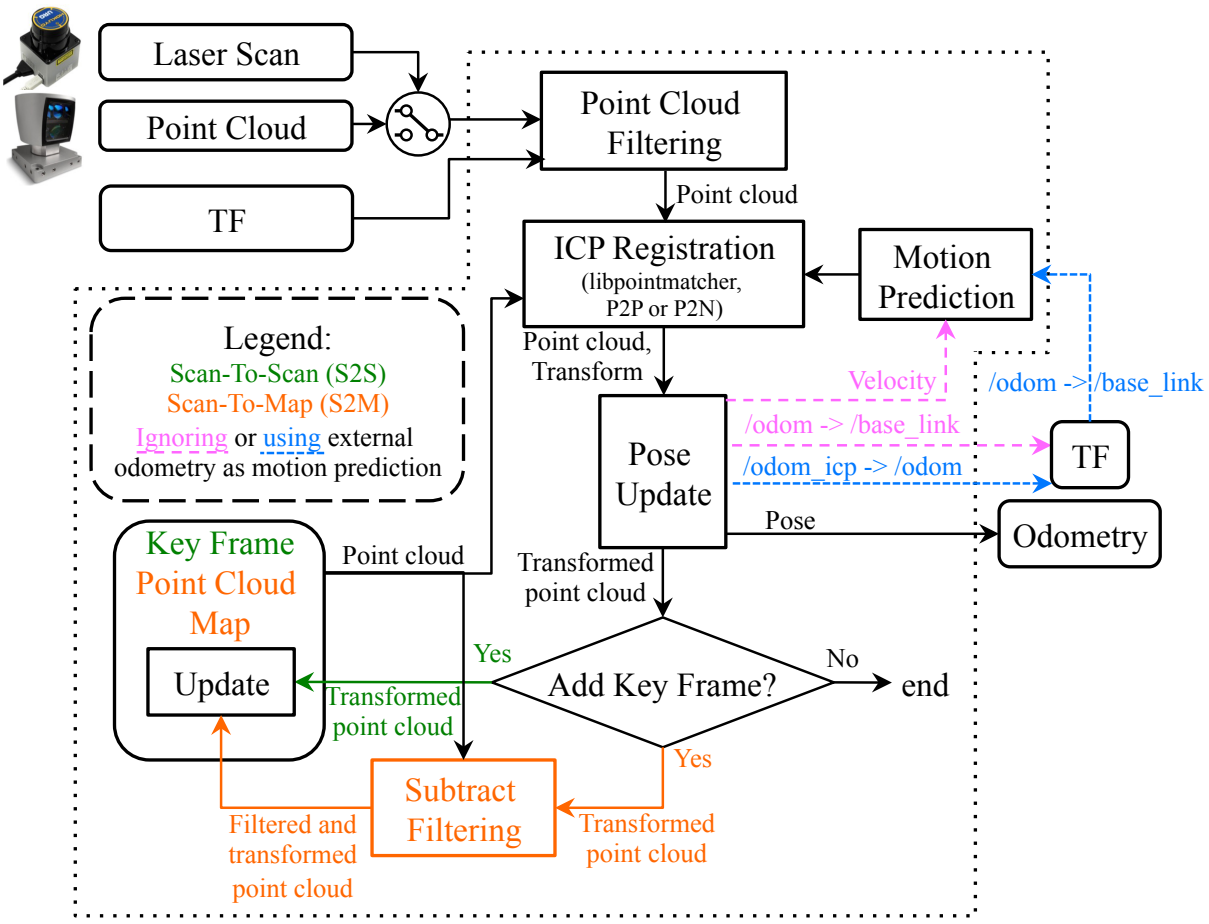


Figure 4.3 Block diagram of *icp_ odometry* ROS node. TF defines the position of the lidar in relation to the base of the robot and as output to publish the odometry transform of the base of the robot. Two odometry approaches can be used: a Scan-To-Scan (S2S) approach in green, and a Scan-To-Map (S2M) approach in orange. The approaches also have the choice of using a constant velocity model (pink) or another source of odometry (blue) for motion prediction. For the later, the correction of the input odometry is published on TF.

– Point Cloud Filtering: The input point cloud is downsampled and normals are computed. *tf* is used to transform the point cloud into robot base frame so that odometry is computed accordingly (e.g., `/base_link`).
– ICP Registration: To register the new point cloud to Point Cloud Map (S2M) or the last Key Frame (S2S), iterative-closest-point (ICP) [Besl et McKay, 1992] is done using implementation of *libpointmatcher* [Pomerleau *et coll.*, 2013]. The Point Cloud Map is a cloud assembled by past key frames. Registration can be done using Point to Point (P2P) or Point to Plane (P2N) correspondences. P2N is preferred in human-made environments with a lot of plane surfaces.

- Motion Prediction: As ICP is dealing with unknown correspondences, this module requires a valid motion prediction before estimating the transform, either from a previous registration or from external odometry approach (e.g., wheel odometry) though *tf* (shown in blue and purple, respectively). Identity transform is provided as motion prediction only at initialization when processing the two first frames. If an external odometry is not used as an initial guess, motion prediction is done according to a constant velocity model based on the previous transformation. A problem with this technique is that if the environment is not complex enough (like in a corridor), odometry may drift a lot if there are no constraints on the direction of the robot. Using an external initial guess in this case can help estimate the motion in the direction in which the environment is lacking features. For example, a robot with a short-range lidar moving in a long corridor in which there are no doors (i.e., not distinguishable geometry) would only see two parallel lines. If the robot accelerates or decelerates in the direction of the corridor, ICP would be able to correct orientation but it would not be able to detect any changes in velocity in the direction of the corridor. In such case, using external odometry can help estimate velocity in the direction in which ICP cannot. If the structural complexity of the current point cloud is lower than the fixed threshold `Icp/PointToPlaneMinComplexity`, only orientation is estimated with ICP and position (along the problematic direction) is derived from the external odometry. The structural complexity of a 2D point cloud is defined as the second Eigenvalue of the Principal Component Analysis (PCA) of the point cloud's normals, multiplied by two. For 3D point cloud, the third Eigenvalue is used multiplied by three.
- Pose Update: After successful registration, odometry pose is then updated. When external odometry is used, *tf* output is the correction of the external odometry *tf* so that both transforms can be in the same *tf* tree (i.e., `/odom_icp`→`/odom`→`/base_link`). Like visual odometry, covariance is computed using the MAD approach [Rusu et Cousins, 2011] between 3D point correspondences.
- Key Frame and Point Cloud Map Update: If the correspondence ratio is under the fixed threshold `Odom/ScanKeyFrameThr`, the new frame becomes the Key Frame for S2S. For S2M, an extra step is done before integrating the new point cloud to Point Cloud Map. The map is subtracted from the new point cloud (using a maximum radius of `OdomF2M/ScanSubtractRadius`), then the remaining points are added to the Point Cloud Map. When the Point Cloud Map has reached the fixed maximum threshold `OdomF2M/ScanMaxSize`, oldest points are removed.

In case ICP cannot find a transformation, odometry is lost. In contrast to visual odometry, lidar odometry cannot recover from being lost when the motion prediction is null, in order to avoid large odometry errors. Lidar odometry must then be reset. However, as long as the lidar can perceive environmental structures, the robot would rarely get lost. Note that if external odometry is used, motion prediction would still give a valid estimation, so ICP registration could recover from being lost if the robot comes back where it has lost tracking. Finally, similarly to the third party visual odometry approaches integrated, an open source version [11] of the lidar odometry approach called LOAM [Zhang et Singh, 2017] has been integrated to RTAB-Map for comparison.

4.3.2 Synchronization

RTAB-Map has a variety of input topics (e.g., RGB-D images, stereo images, odometry, 2D laser scan, 3D point cloud and user data) that can be used depending on the sensors available. The minimum topics that are required to make the *rtabmap* ROS node work are registered RGB-D or calibrated stereo images with odometry, provided through a topic or by *tf* (e.g., `/odom`→`/base_link`). RTAB-Map also supports multiple RGB-D cameras as long as they have all the same image size. Accurate *tf* of the sensors used are required (e.g., `/base_link`→`/camera_link`). Figure 4.4 and Figure 4.5 illustrate two visual SLAM examples with corresponding *tf* trees. RTAB-Map's visual odometry nodes can be replaced by any other odometry approaches (i.e., wheel odometry, other visual odometry package, lidar odometry, etc.). The dotted links show which node is publishing the corresponding *tf*. For other *tf* frames describing the position of the sensors on the robot, they are usually published by the camera driver, by some *static_transform_publisher* [12] or by a *robot_state_publisher* [13] using Unified Robot Description Format of the robot.

Once subscribed to basic sensors, there are two other topics that can be optionally synchronized: a 2D laser scan (e.g., Hokuyo and SICK lidars) or a 3D point cloud (e.g., Velodyne lidar) to generate 2D and 3D occupancy grids, respectively. They can be also used to refine the links in the graph using ICP.

As sensors do not always publish data at the same rate and at the same exact time, good synchronization is important to avoid bad registration of the data. ROS provides two kind of synchronization: exact and approximate. An exact synchronization requires that input topics have exactly the same timestamp, i.e., for topics coming from the same sensor (e.g., left and right images of a stereo camera). An approximate synchronization compares

11. `https://github.com/laboshinl/loam_velodyne`
12. `http://wiki.ros.org/tf#static_transform_publisher`
13. `http://wiki.ros.org/robot_state_publisher`

timestamps of the incoming topics and try to synchronize all topics with a minimum delay error. It is used for topics coming from different sensors. Synchronization can then become a little tricky if a subset of input topics (i.e., camera topics) must be synchronized with the exact time policy while being approximately synchronized with other sensors. To do so, the *rtabmap ros/rgbd_ sync* ROS nodelet can be used to synchronize camera topics into a single topic of type *rtabmap_ ros/RGBDImage*[14] before the *rtabmap* node. Figure 4.6 presents a synchronization example with a RGB-D camera and a lidar. For RGB-D camera, ROS packages do not always provide exact same timestamps for RGB and depth images, and *rgbd_ sync* can be also used with approximate synchronization to synchronize images at the camera frame rate (e.g., 30 Hz) independently of the rate of the other inputs (e.g., laser scan, odometry).

4.3.3 STM

When a new node is created in STM, in complement to the information described in [Labbé et Michaud, 2017], a local occupancy grid is now computed from the depth image, the laser scan or the point cloud. In case of stereo images, a dense disparity image is computed using a block matching algorithm [Konolige, 1998], and converted to a point cloud. A local occupancy grid is referenced in the robot frame and it contains empty, ground and obstacle cells at the fixed grid cell size `Grid/CellSize`. The total size of a local occupancy grid is defined by the range and field of view of the sensor used to create it. These local occupancy grids are used to generate a global occupancy grid by transforming them in map referential using the poses of the map's graph. While pre-computing the local occupancy grids requires more memory for each node, it greatly decreases the regeneration time of the global occupancy grid when the graph has been optimized. For example, in previous work [Labbé et Michaud, 2014], when the global occupancy grid had to be generated, all the laser scans had to be ray traced again.

Depending on the parameters `Grid/FromDepth`, `Grid/3D` and the input topics set, the local occupancy grid is generated differently and the result is either 2D or 3D, as shown in Figure 4.7. For example, if parameter `Grid/FromDepth` is false and *rtabmap* node is subscribed to a laser scan topic, a local 2D occupancy grid is created. A 2D local occupancy grid requires less memory than the 3D one because there is one less dimension to save (e.g., z) and superposed obstacles can be reduced to only one obstacle cell. However, local 2D occupancy grids cannot be used to generate a 3D global occupancy grid, while local 3D occupancy grids can be used to generate 2D and 3D global occupancy grids. The choice

14. `http://docs.ros.org/api/rtabmap_ros/html/msg/RGBDImage.html`

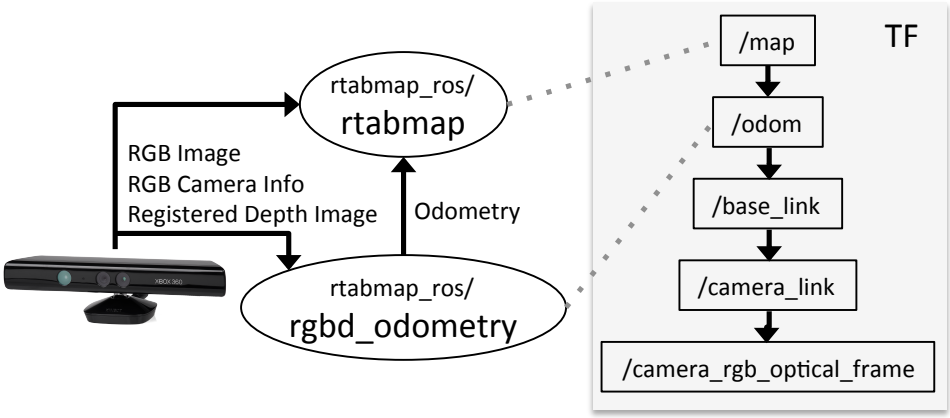


Figure 4.4 Visual SLAM with a RGB-D camera like the Kinect for Xbox 360. The *rgbd_odometry* ROS node is used to compute odometry for *rtabmap* ROS node. On the right is a standard resulting TF tree for this sensor configuration (with transforms linked by a dotted line to corresponding publishing ROS nodes).

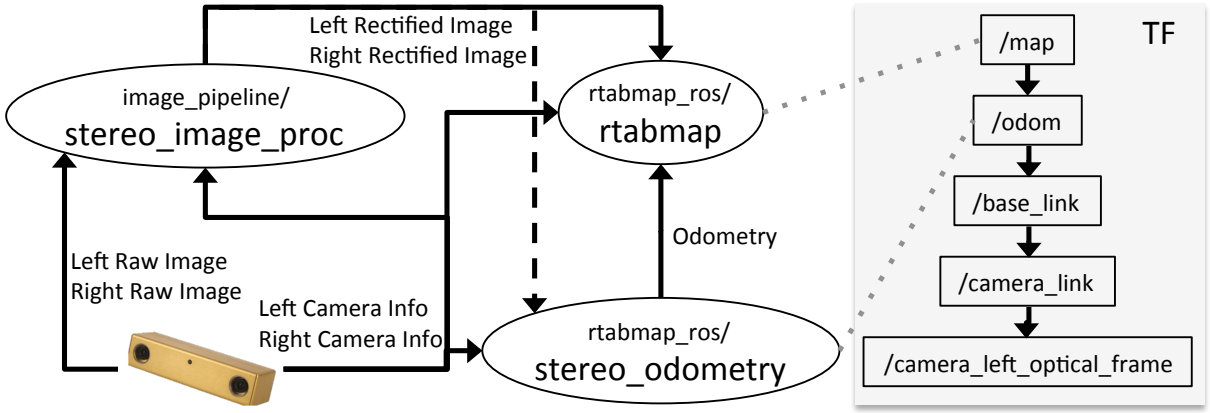


Figure 4.5 Visual SLAM with a stereo camera like the BumbleBee2. The *stereo_odometry* ROS node is used to compute odometry for *rtabmap* ROS node. RTAB-Map's ROS nodes require rectified stereo images, thus the standard *stereo_image_proc* ROS node is used to rectify them. On the right is a standard resulting TF tree for this sensor configuration (with transforms linked by a dotted line to corresponding publishing ROS nodes).

depends on what kind of global map is required for the application and on the processing power available. Note that if `Grid/FromDepth` is false and no laser scan and point cloud topics are subscribed, no grids are computed. The rectangular boxes in Figure 4.7 are described as follows:

– 2D Ray Tracing: For each ray of the laser rangefinder, a line is traced on the grid to fill empty cells between the sensor and the obstacle hit by the ray. It is assumed that

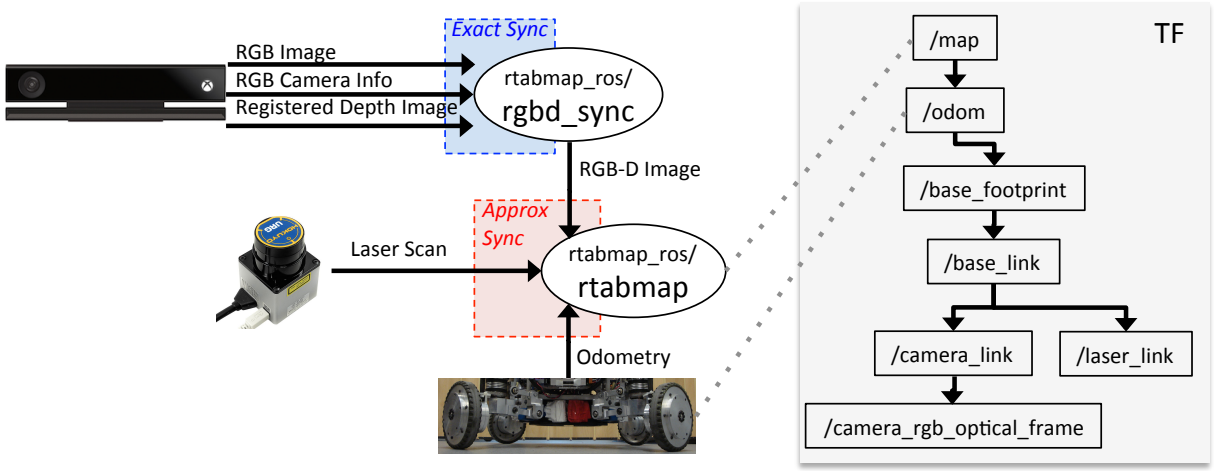


Figure 4.6 Synchronization example of a RGB-D camera (Kinect for Xbox One) with laser scan (URG-04LX) and odometry. In this case, odometry is computed through wheel encoders. Camera messages are synchronized together using *rgbd_sync* ROS node before synchronizing the resulting RGB-D image message with the other sensors (which can have different publishing rates). On the right is an example of the resulting TF tree for this sensor configuration (with transforms linked by a dotted line to corresponding publishing ROS nodes).

the rays are parallel to the ground. This approach can generate 2D local occupancy grids very fast and is done by default for 2D lidar-based mapping.

- Depth Image to Point Cloud: The input depth image (or disparity image in case of stereo images) is projected in 3D space according to sensor frame and camera calibration. The cloud is then transformed in the robot base frame.
- Filtering and Ground Segmentation: The point cloud is downsampled by a voxel grid filter [Rusu et Cousins, 2011] with voxel size equals to fixed grid cell size. The ground plane is then segmented from the point cloud: the normals of the point cloud are computed, then all points with their normal parallel to z-axis (upward) within the fixed maximum angle `Grid/MaxGroundAngle` are libelled as ground, others are obstacles.
- Projection: If `Grid/3D` is false, the 3D ground and obstacle point clouds are projected on ground plane (e.g., x-y plane). The voxel grid filter is applied again to merge points projected in the same cell. 2D ray tracing can be done to fill empty space between obstacles and the camera. If 2D ray tracing is not used and if the point cloud does not have any points segmented as ground, no empty cells are set in the occupancy grid between the sensor and the obstacles.
- 3D Ray Tracing: An OctoMap is created from the single local occupancy grid in the robot referential. OctoMap does 3D ray tracing and detects empty cells between the

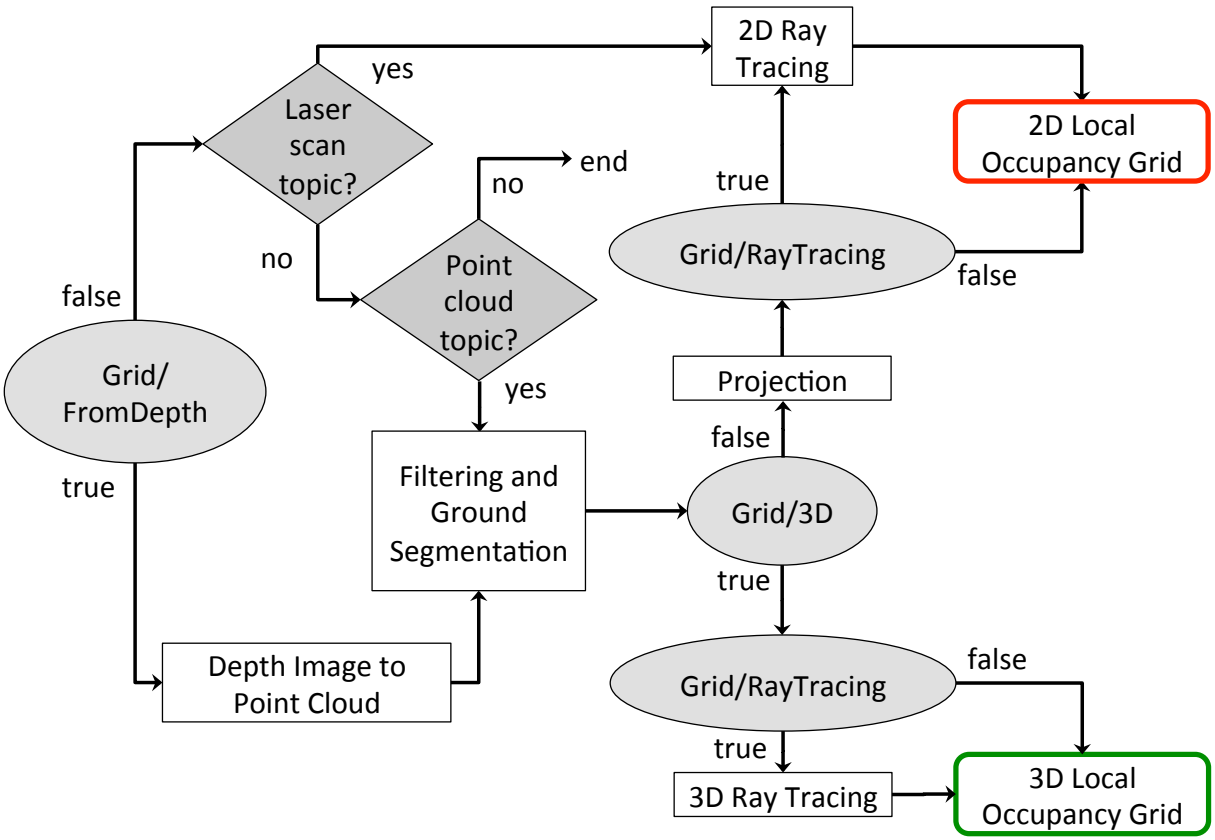


Figure 4.7 STM's local occupancy grid creation. Depending on the parameters (shown by ellipses) and the availability of the optional laser scan and point cloud inputs (shown by diamonds), the local occupancy grid can either be 2D or 3D.

camera and occupied cells. The OctoMap is converted back to local occupancy grid format with empty, ground and obstacle cells.

4.3.4 Loop Closure and Proximity Detection

Loop closure detection is done using the bag-of-words approach described in [Labbé et Michaud, 2013]. Basically, when creating a new node, STM extracts visual features from the RGB image and quantizes them to an incremental visual word vocabulary. Features can be any of the types included in OpenCV like SURF [Bay *et coll.*, 2008], SIFT [Lowe, 2004], ORB [Rublee *et coll.*, 2011] or BRIEF [Calonder *et coll.*, 2010]. When visual odometry F2F or F2M is used, it is possible to re-use features already extracted for odometry for loop closure detection. This eliminates extracting twice the same features. As loop closure detection does not need as many features than odometry to detect loop closures and to reduce computation load, only a subset (maximum of `Kp/MaxFeatures`) of the odometry features with highest response are quantized to visual word vocabulary. The other features are still kept in the node when loop closure transformation has to be

computed. The created node is then compared to nodes in WM to detect a loop closure. STM contains the last nodes added to map, and therefore these nodes are not used for loop closure detection. Locations in STM would be very similar to last location and would bias loop closure hypotheses on them. STM can be seen as a buffer of a fixed size `Mem/STMSize` before a node is moved WM. To compute likelihood between the created node and all those in WM, Tf-IDF approach [Sivic et Zisserman, 2003] is used to update a Bayes filter estimating the loop closure hypotheses. The filter estimates if the new node is from a previously visited location or a new location. When a loop closure hypothesis reaches the fixed threshold `Rtabmap/LoopThr`, a loop closure is detected and transformation is computed. The transformation is computed using the same Motion Estimation approach used by visual odometry (Section 4.3.1), and if accepted, the new link is added to the graph. When a laser scan or a point cloud is available, link's transformation is refined using the same ICP Registration approach than with lidar odometry (described in Section 4.3.1).

Introduced in [Labbé et Michaud, 2017], proximity detection is used to localize nodes close to the current position with laser scans (when available). For example, with proximity detection, it is possible to do localization when traversing back the same corridor in a different direction, during which the camera can not be used to find loop closures. In contrast to loop closure detection where complexity depends on the WM size, the complexity of proximity detection is bounded to nodes close to the robot. These nodes must be close in the graph, i.e., the number of links between them and the latest node should be less than the fixed threshold `RGBD/ProximityMaxGraphDepth`. When odometry drifts over large distance, the robot may move to a previously mapped area that differs from the current real location, so using this threshold, proximity detection do not make comparison with nodes from the previously mapped area to avoid invalid proximity detections. If odometry does not drift too much or that the map update rate is higher, the threshold can be set higher, otherwise it should be lowered.

4.3.5 Graph Optimization

When a loop closure or a proximity detection are detected or some nodes are retrieved or transferred because of memory management, a graph optimization approach is applied to minimize errors in the map. RTAB-Map integrates three graph optimization approaches: TORO [Grisetti *et coll.*, 2010], g2o [Kummerle *et coll.*, 2011] and GTSAM [Dellaert, 2012]. g2o and GTSAM converge faster than TORO, but are less robust to multi-session mapping when multiple independent graphes have to be merged together. TORO is also less

sensitive to poorly estimated odometry covariance. However, for single map, based on empirical data, g2o and GTSAM optimization quality is better than TORO, particularly for 6DoF maps. GTSAM is slightly more robust to multi-session than g2o, and thus is the strategy now used by default in RTAB-Map contrarily to our previous works using TORO.

Visual loop closure detection is not error-free, and very similar places can trigger invalid loop closure detections, which would add more errors to the map rather than reducing them. To detect invalid loop closure or proximity detections, RTAB-Map now uses a new parameter. If a link's transformation in the graph after optimization has changed more than than the factor `RGBD/OptimizeMaxError` of its translational variance, all loop closure and proximity links added by the new node are rejected, keeping the optimized graph as if no loop closure happened.

4.3.6 Global Map Assembling

Figure 4.8 illustrates the global map outputs that can be assembled from the local occupancy grids of Figure 4.7. Saving 3D local occupancy grids in nodes gives to most flexibility, as they can be used to generate all types of map. However, if only a 2D global occupancy grid map is needed, saving already projected local grids in the nodes saves memory (two numbers per point instead of three) and time (points are already projected to 2D) when assembling the local maps. Using the map's graph, each local occupancy grid are transformed into its corresponding pose. When a new node is added to map, the new local occupancy grid is combined with the global occupancy grid, clearing and adding obstacles. When a loop closure occurs, the global map should be re-assembled according to all new optimized poses for all nodes in the map's graph. This process is required so that obstacles that have been incorrectly cleared before the loop closure can be reincluded. The point cloud outputs assemble all points of the local maps and publish them in the standard *sensor_ msgs/PointCloud2*[15] ROS format. Voxel grid filtering is done to merge overlapping surfaces. The resulting point cloud is a convenient format for visualization and debugging, and ease integration with third party applications.

15. `http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html`

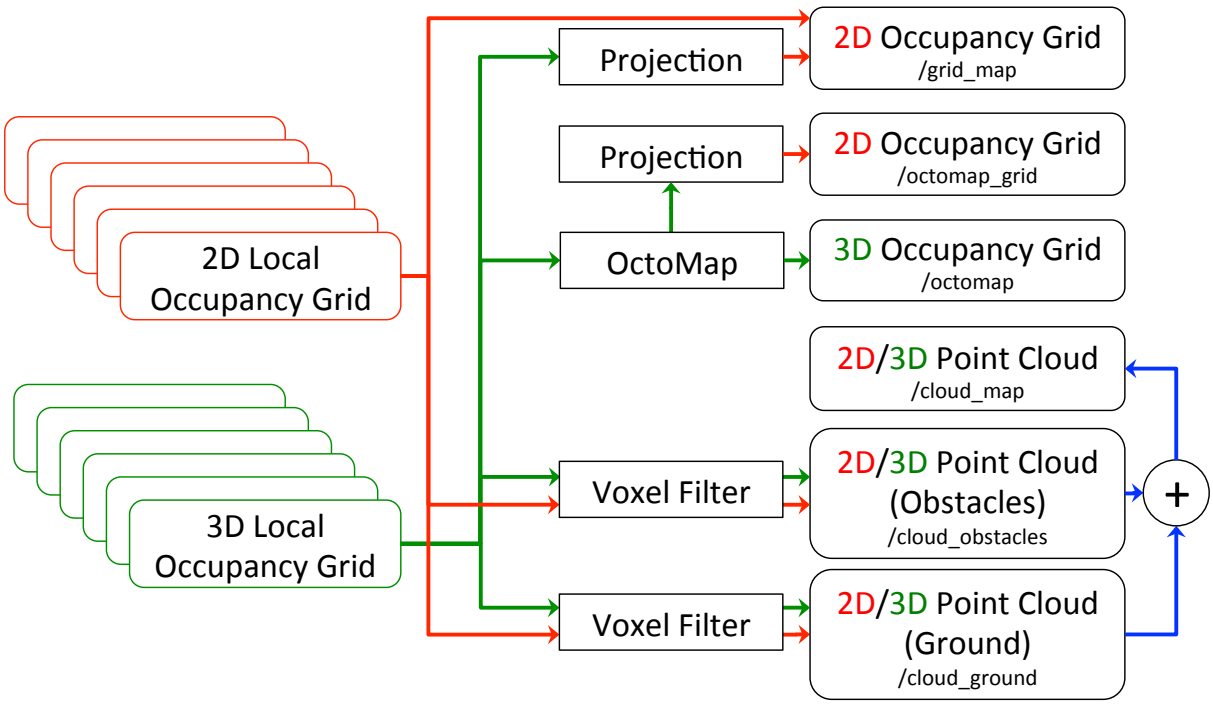


Figure 4.8 Global map assembling. Depending on the type of local maps created in the map's graph (see Figure 4.7), the available output global maps will differ. Only 3D local occupancy grids can be used to generate the 3D occupancy grid (OctoMap) and its projection in 2D.

4.4 Evaluating Trajectory Performance of RTAB-Map Using Different Sensor Configurations

Performance of RTAB-Map has been evaluated on four datasets having a ground truth: KITTI [Geiger *et coll.*, 2012], TUM RGB-D [Sturm *et coll.*, 2012], EuRoC [Burri *et coll.*, 2016] and PR2 MIT Stata Center [Fallon *et coll.*, 2013]. These datasets have a variety of sensors (i.e., stereo and RGB-D cameras, 2D and 3D lidars, combined wheel and IMU odometry). Using RTAB-Map as the common evaluation framework makes it possible to outline performance differences between all sensor and odometry configurations. The metric used for trajectory accuracy is the absolute trajectory (ATE) root-mean-square error, derived from the TUM RGB-D benchmark [Sturm *et coll.*, 2012]. All errors are computed with the map's graph including loop closures.

Experiments were conducted on a desktop computer using an Intel® Core™ i7-3770 Processor (four cores), 6 GB of RAM and 512 GB SSD running Ubuntu 16.04. To compare trajectory accuracy of the different SLAM configurations independently of the computation power available, KITTI, TUM and EuRoC datasets have been processed offline, so

Table 4.2 RTAB-Map (version 0.16.3) Default Parameters

GFTT/MinDistance	3 pixels	RGBD/OptimizeMaxError	1
GFTT/QualityLevel	0.001	RGBD/ProximityMaxGraphDepth	50 nodes
Kp/MaxFeatures	500 features	Rtabmap/DetectionRate	2 Hz
Odom/KeyFrameThr (F2M)	0.3	Rtabmap/TimeThr	0 ms
Odom/KeyFrameThr (F2F)	0.6	Rtabmap/MemoryThr	0 nodes
Odom/ScanKeyFrameThr	0.9	Rtabmap/LoopThr	0.11
OdomF2M/MaxSize	2000 features	Vis/CorNNDR	0.6
OdomF2M/ScanMaxSize	10000 points	Vis/CorGuessWinSize	20 pixels
OdomF2M/ScanSubtractRadius	0.05	Vis/MaxFeatures	1000 features
Mem/RehearsalSimilarity	0.2	Vis/MinInliers	20
Mem/STMSize	30 nodes		

that every frames are processed by odometry even if in some cases odometry processing time is higher than the camera frame rate. To compare the overall computation load in relation to computation time, a single core is used for offline experiments. For the PR2 MIT Stata Center dataset, experiments were conducted online in ROS and are not limited to a particular core of the computer. Table 4.2 shows default parameters (unless explicitly specified) used for all datasets. To provide a fair comparison between results from RTAB-Map and other SLAM approaches, RTAB-Map's memory management has been disabled (Rtabmap/TimeTheshold and Rtabmap/MemThreshold set both to 0) in these experiments.

With the offline datasets, the different stereo odometry approaches tested are using their own stereo correspondence approaches, which can generate slightly different disparity values. We also observed that camera calibration are not always accurate: rectified images still contain some visible distortions. These problems can influence the scale of the created map, thus bias trajectory accuracy when comparing with the ground truth (for which we assume there is no scale error). As observed in [Mur-Artal et Tardós, 2017], depth image values from some TUM RGB-D sequences can be slightly off, causing also a scale problem. For a fair comparison between all visual odometry approaches, as we cannot know if the scale problem is caused by the camera calibration and/or the disparity computation approach, results are presented with the map scaled to minimize the error against the ground truth. When ORB-SLAM2 is used as odometry input to RTAB-Map, it is referred to as ORB2-RTAB to differentiate from results of ORB-SLAM2 full SLAM version. Similarly, when LOAM is used as odometry input to RTAB-Map, it is referred to as LOAM-RTAB.

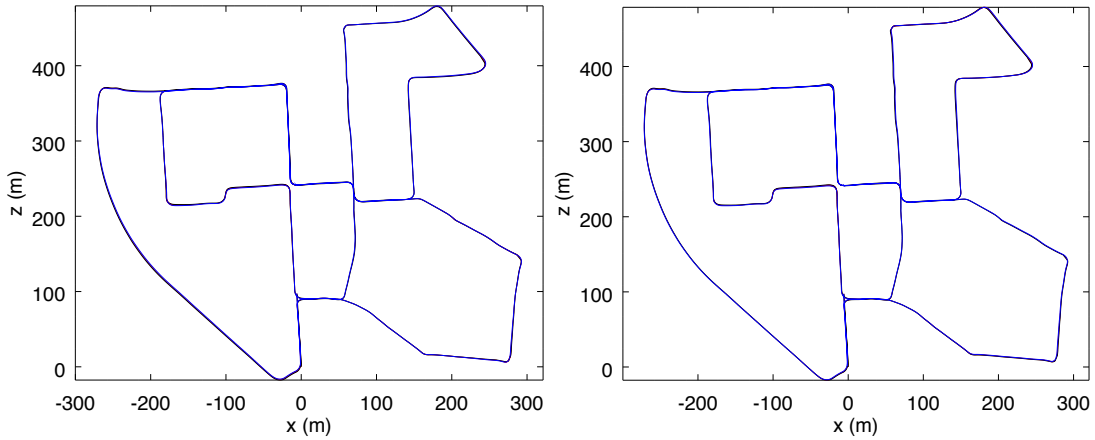
Note that the focus of the paper is on pose estimation evaluation to compare SLAM approaches. Localization robustness is not addressed explicitly but in all the results presented, no wrong loop closures were accepted. For similar places, either they were rejected by the criteria of not having enough visual inliers (see Motion Estimation of Section 4.3.1)

or because of large graph optimization errors (see Section 4.3.5), making the evaluations robust to invalid localization for the datasets tested.

4.4.1 KITTI

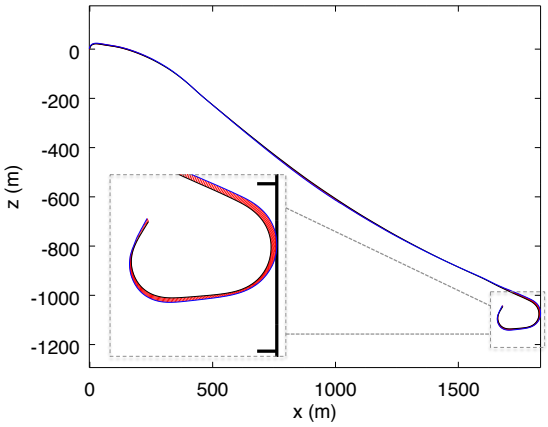
In the KITTI dataset, stereo images were recorded from two synchronized monochrome PointGrey cameras installed on the top of a car. The dataset provides rectified stereo images of size 1241×376 pixels with baseline of 0.54 m at 10 Hz. The dataset contains also 3D point clouds coming from a Velodyne 64E installed on the top of the car. The point clouds are synchronized with stereo images at 10 Hz. For S2M and S2S approaches using the lidar, to reduce computation load and memory usage, the raw point clouds are downsampled using a voxel filter of 50 cm in the Point Cloud Filtering step. Based on preliminary tests, Point-to-Point ICP Registration has been chosen because it gives slightly better results than Point-to-Plane on this dataset, in particular in sequences where there are lot of trees and not many planes. For S2M, `OdomF2M/ScanSubtractRadius` is set to 50 cm to match the voxel filter. As Velodyne data is 360° with far range, `Odom/ScanKeyFrameThr` is set to 0.8 instead of 0.9 to trigger new key frames less often. In comparison to other datasets, KITTI has larger images. Therefore, parameters affected by image size are modified: `GFTT/MinDistance` is set to 7, `GFTT/QualityLevel` is set to 0.01, `Vis/MaxFeatures` is set to 1500, `Kp/MaxFeatures` is set accordingly to half of `Vis/MaxFeatures` at 750 and `OdomF2M/MaxSize` is set to 3000.

Figure 4.9 presents a comparison of visual and lidar trajectories derived using RTAB-Map with stereo odometry (F2M) and lidar odometry (S2S), in relation to ground truths of three sequences in the KITTI dataset: sequence 00 has loop closures; sequence 01 has been taken on a highway at a speed of 90 km/h and higher; sequence 08 does not have loop closures (e.g., the camera is oriented in a different direction when traversing back the same segments). When comparing the trajectories at that scale, there is not so much difference between visual and lidar approaches and both follow well the ground truth. Table 4.3 summarizes trajectory accuracy in terms of ATE for all odometry configurations available in RTAB-Map, along with performance reported for ORB-SLAM2 [Mur-Artal et Tardós, 2017], LSD-SLAM [Engel *et coll.*, 2015] and SOFT-SLAM [Cvišić *et coll.*, 2018]. o_{avg} is the average odometry time across all sequences when limiting the approach to a single CPU Core. For three sequences (06, 07 and 09), using the Velodyne provides better performance compared to the visual-based approaches. However, for sequence 01, which is a highway sequence not geometrically complex, lidar-based approaches perform quite worst: a lot of error along y axis (in KITTI coordinates) are caused by bad pitch

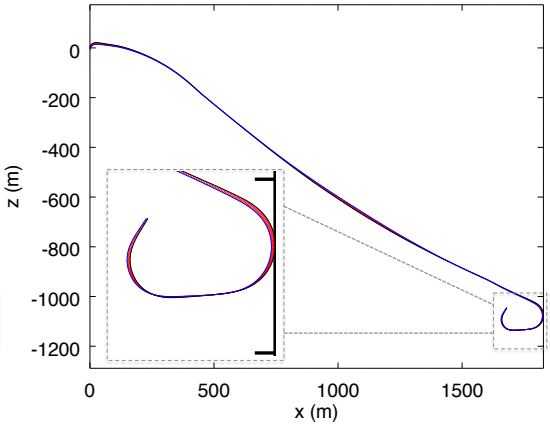


(a) 00 - F2M

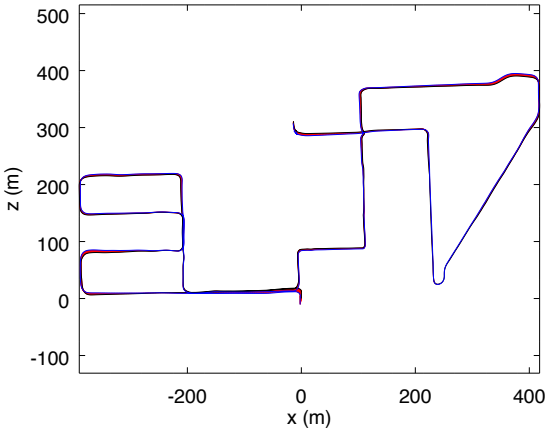
(b) 00 - S2S



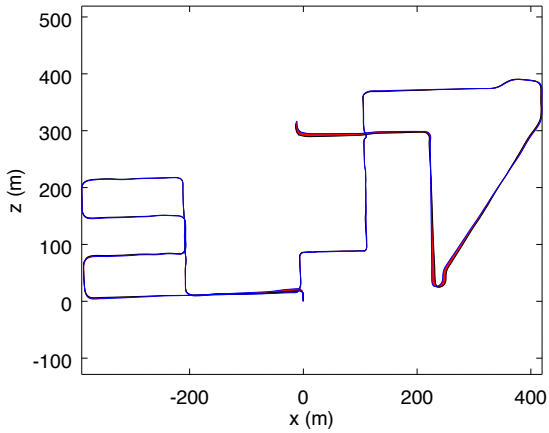
(c) 01 - F2M



(d) 01 - S2S



(e) 08 - F2M



(f) 08 - S2S

Figure 4.9 Trajectories using RTAB-Map with stereo odometry F2M (left) and lidar odometry S2S (right) against ground truths for the three KITTI sequences 00, 01 and 08. Errors between poses estimated by RTAB-Map (blue) and the ground truths (black) are shown in red.

Table 4.3 ATE (m) results for the KITTI sequences in relation to the odometry approach and the sensor used using a single CPU core

Sensor	RTAB-Map Odometry	KITTI Sequence 00	01	02	03	04	05	06	07	08	09	10	o_{avg} (msec)
Lidar	S2S	**1.0**	24.0	3.1	0.7	0.4	0.6	**0.5**	**0.3**	4.2	**1.1**	1.8	62
	S2M	1.1	17.2	2.9	0.7	0.5	0.7	**0.5**	**0.3**	7.7	**1.1**	1.7	82
	LOAM-RTAB	1.8	23.3	47	1.1	0.3	1.1	1	0.4	10.1	1.3	3	330
Stereo Camera	F2F	1.4	14.5	4.7	0.4	**0.2**	0.72	1.8	0.6	5.8	2.2	3.0	61
	F2M	**1.0**	4.7	4.7	0.3	**0.2**	**0.5**	0.8	0.5	3.8	2.8	**0.8**	82
	Fovis	28.8	×	×	2.0	1.9	11.4	24.5	3.8	25	33.3	13.6	**17**
	ORB2-RTAB	**1.0**	5.3	4.4	**0.2**	**0.2**	**0.5**	0.6	0.5	3.0	1.5	0.9	175
	Viso2	4.2	19.7	37.9	1.2	0.3	3.4	6.3	1.1	22.4	4.1	4.8	66
LSD-SLAM (stereo)		**1.0**	9.0	**2.6**	1.2	**0.2**	1.5	1.3	0.5	3.9	5.6	1.5	-
ORB-SLAM2 (stereo)		1.3	10.4	5.7	0.6	**0.2**	0.8	0.8	0.5	3.6	3.2	1.0	-
SOFT-SLAM (stereo)		1.2	**3**	5.1	0.5	0.4	0.8	**0.5**	**0.3**	**2.3**	1.3	0.9	-

estimation. In contrast, visual-based approaches can use features farther to lidar range to better estimate the pitch orientation. S2M and S2S give relatively the same errors, so choosing between the two could be based on computation time o_{avg}, with S2S always being the fastest. LOAM-RTAB is better only for the fourth sequence in comparison to other lidar-based approaches. Comparing only visual-based approaches, ORB2-RTAB is the best for nine of the 11 sequences and F2M in six sequences (with ties for four sequences). However, ORB2-RTAB cannot satisfy real-time constraints (o_{avg} is always over 100 ms). In the ORB-SLAM2 paper [Mur-Artal et Tardós, 2017], the sequences can be processed under 100 ms using multiple cores, while here only a single core is used. On less powerful computers using a stereo camera, F2M odometry seems the better choice. Fovis is the fastest configuration, but it gets lost quite easily: there are not enough visual inliers to compute transformation, and odometry is automatically reset inside its library, causing some missing motions in the map or very bad transformations (where "×" means a very high drift).

To facilitate comparison with other papers using the KITTI dataset, Table 4.4 presents the average translational error metric reported with the KITTI dataset [Geiger *et coll.*, 2012]. Results of LOAM [Zhang et Singh, 2017], a lidar-based approach, are also shown. Translational error (in percent) have been computed for all possible subsequences of length (100 m, ..., 800 m) of each sequence, then averaged together. ORB2-RTAB approach performs best in 10 out of 11 sequences. The difference between ORB2-RTAB and ORB-SLAM2 can be explained by the difference between the loop closure detection and graph optimization approaches, and also by the map scaling process explained at the beginning of Section 4.4. LOAM, using a more complex scan matching approach, scores better on seven out of the 11 sequences when compared only to lidar odometry approaches. This suggests that

Table 4.4 Average translational error (%) results for the KITTI sequences in relation to the odometry approach and the sensor used

Sensor	RTAB-Map Odometry	KITTI Sequence 00	01	02	03	04	05	06	07	08	09	10
Lidar	S2S	0.82	3.17	1.26	1.02	1.21	0.51	0.58	0.58	1.11	0.90	1.64
	S2M	0.86	2.52	1.14	1.03	1.18	0.56	0.58	0.65	1.25	0.90	1.52
	LOAM-RTAB	1.2	2.9	4.4	1.1	1.5	0.8	0.9	0.6	1.5	1.2	1.7
Stereo Camera	F2F	0.85	2.38	1.01	0.90	**0.35**	0.49	1.25	0.62	1.56	1.24	1.71
	F2M	0.68	2.04	0.97	0.77	0.45	0.38	0.57	0.56	1.17	1.38	**0.48**
	Fovis	9.09	×	×	1.79	2.22	4.26	6.95	3.65	5.39	14.8	10.6
	ORB2-RTAB	0.67	0.96	**0.75**	**0.62**	0.50	**0.35**	0.48	0.53	1.06	0.87	0.54
	Viso2	2.38	5.92	4.19	1.94	0.66	1.85	4.60	1.04	2.82	1.68	1.93
LOAM (lidar)		0.78	1.43	0.92	0.86	0.71	0.57	0.65	0.63	1.12	0.77	0.79
LSD-SLAM (stereo)		**0.63**	2.36	0.79	1.01	0.38	0.64	0.71	0.56	1.11	1.14	0.72
ORB-SLAM2 (stereo)		0.70	1.39	0.76	0.71	0.48	0.40	0.51	0.50	1.05	0.87	0.60
SOFT-SLAM (stereo)		0.66	**0.86**	1.36	0.70	0.50	0.43	**0.41**	**0.36**	**0.78**	**0.59**	0.68

extracting geometric features from the point clouds can help get better motion estimation than by using ICP only. When comparing LOAM-RTAB and LOAM against each other, either the open source version used (with default parameters for this kind of Velodyne) seems to not reproduce the original algorithm perfectly, or some parameters tuning is required, as we expected that results would be more similar. Finally, results of RTAB-Map with F2M odometry approach have been submitted to KITTI's odometry benchmark [16] for the testing sequences 11 to 21. Table 4.5 presents a snapshot of the current ranking with other approaches in Table 4.3. Compared to popular ORB-SLAM2 and LSD-SLAM approaches, RTAB-Map's translation error is very close, even slightly better in terms of rotation performance. Note that LOAM, SOFT-SLAM and LSD-SLAM (stereo version) are not currently available as a C++ library or with ROS, which make them difficult to use on a real robot.

4.4.2 TUM

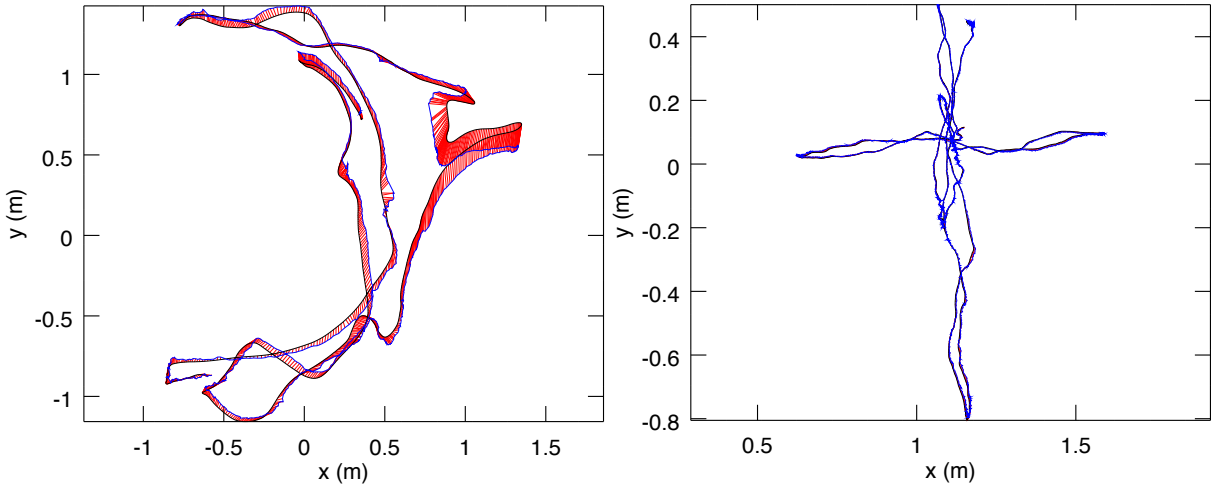
The TUM RGB-D dataset was recorded using a handheld Kinect v1 in small office-like environments. RGB and depth Images were recorded at 30 Hz and are synchronized using the tool provided with the dataset.

Figure 4.10 illustrates the trajectories for F2M compared to ground truths for three TUM sequences. In the fr1 sequence, the camera is moving and rotating faster than in other sequences, resulting in an estimated trajectory diverging more from the ground truth. When moving fast with this kind of camera, synchronization between RGB and depth images is poor (i.e., RGB pixels do not always match with the right depth pixels), causing

16. `http://www.cvlibs.net/datasets/kitti/eval_odometry.php`

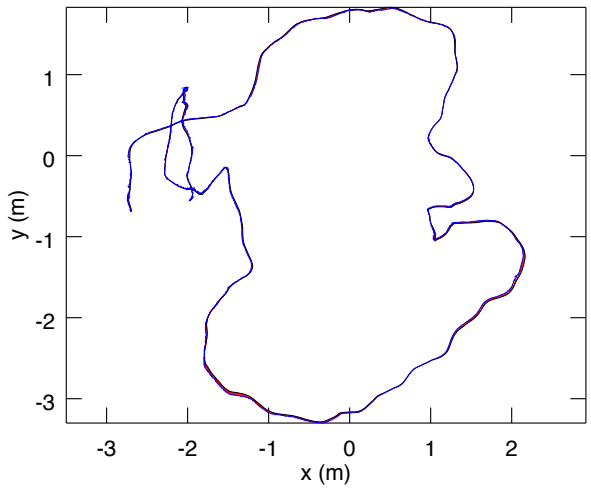
Table 4.5 Current state of KITTI's odometry leaderboard

Rank	Method	Setting	Translation	Rotation
2	LOAM	Lidar	0.61 %	0.0014 [deg/m]
5	SOFT-SLAM	Stereo	0.65 %	0.0014 [deg/m]
30	ORB-SLAM2	Stereo	1.15 %	0.0027 [deg/m]
38	LSD-SLAM	Stereo	1.20 %	0.0033 [deg/m]
47	RTAB-Map (F2M)	Stereo	1.26 %	0.0026 [deg/m]
73	Viso2	Stereo	2.44 %	0.0114 [deg/m]



(a) fr1-room

(b) fr2-xyz



(c) fr3-office

Figure 4.10 Trajectories using RTAB-Map with RGB-D odometry F2M (blue) against ground truths (black) for three TUM sequences. Errors between poses estimated by RTAB-Map and the ground truth are shown in red.

bad motion estimations. Table 4.6 presents ATE results with additional ones from other approaches like Elastic Fusion [Whelan *et coll.*, 2016], Kintinuous [Whelan *et coll.*, 2015],

DVO SLAM [Kerl *et coll.*, 2013], RGBDSLAMv2 [Endres *et coll.*, 2014], RGBiD-SLAM [Gutierrez-Gomez *et coll.*, 2016] BundleFusion[Dai *et coll.*, 2017] for comparison. Comparing RTAB-Map approaches together, ORB2-RTAB scores best in six out of the seven TUM sequences, followed by F2M. Higher errors observed with ORB2-RTAB compared to the original ORB-SLAM2 are because in the ORB-SLAM2 approach a global bundle adjustment is done after a loop closure: as there are a lot of visual features shared between many key frames, global bundle adjustment can indeed provide, with more computation time, better optimization than only optimizing the links between the graph's nodes as in RTAB-Map. In comparison to RTAB-Map's odometry approaches F2F and F2M, ORB2-RTAB seems less sensitive to large depth error at distance greater than 4 m with this kind of sensor, and to bad synchronization between RGB and depth cameras. In other words, ORB-SLAM2 refines the 3D position of the features in the feature map when new frames arrive, providing better triangulation of the features even if the initial depth taken form the depth image is erroneous. This could explain why ORB2-RTAB outperforms F2F and F2M. F2M still perform well in comparison to other visual SLAM approaches. Also, because of the fast rotation motions in fr1 sequences, F2F has problems tracking the features with optical flow in comparison with feature matching used by F2M. Finally, Fovis is the fastest and the only real-time odometry approach (under 33 ms), followed by F2F and DVO.

4.4.3 EuRoC

The EuRoC dataset has 11 stereo image sequences at 20 Hz taken on a drone in small indoor rooms (V1 and V2) and in a machine room (MH). Synchronized IMU data with camera are also available. While the images are time synchronized, exposure level between the cameras is not (e.g., right image can be darker with lower contrast than the left one). This increases the difficulty to find good stereo correspondences between left and right images when computing the disparity. To mitigate this problem, exposure compensation [Xu et Mulligan, 2010] is done between left and right images before processing them by the odometry approaches. For OKVIS odometry, the IMU is used along the stereo images.

Figure 4.11 shows the paths computed for three EuRoC sequences using a stereo odometry approach (F2M) and the visual inertial odometry approach of OKVIS, compared against the ground truths. Except for the V2-03-difficult sequence where F2M fails to estimate the whole trajectory, the results between stereo visual odometry and visual inertial odometry are similar. Table 4.7 presents ATE results for all sequences. Overall, ORB2-RTAB performs better on six of the 11 sequences when compared to other RTAB-Map's odometry

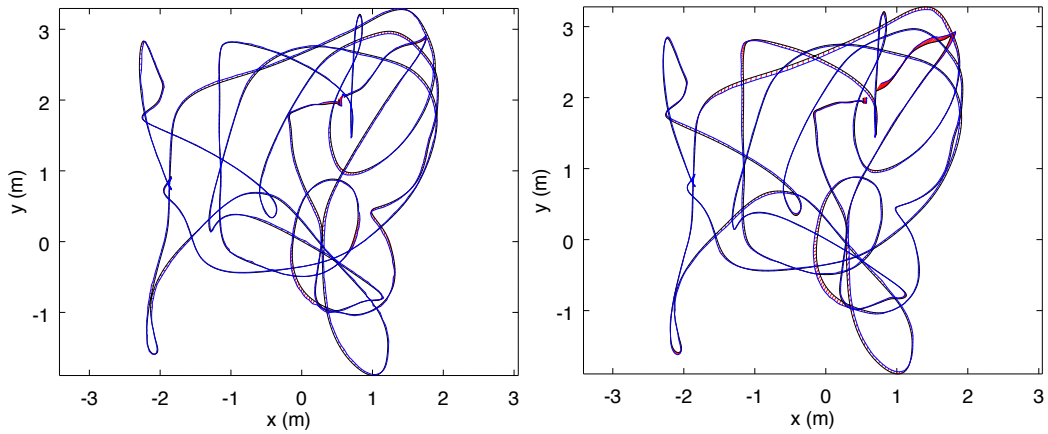
Table 4.6 ATE (cm) results for the TUM sequences in relation to the odometry approach

RTAB-Map	TUM fr1			TUM fr2		TUM fr3		o_{avg}
Odometry	desk	desk2	room	desk	xyz	office	nst	(msec)
F2F	7.2	10.1	8.8	2.2	0.5	2.6	7.4	37
F2M	2.9	4.4	6.6	2.4	0.5	2.1	1.7	70
DVO	5.9	6.7	10.7	6.0	0.8	10.8	3.5	37
Fovis	4.8	8.8	11.9	4.7	0.7	5.1	10.6	**21**
ORB2-RTAB	1.9	4.3	10.3	1.2	**0.4**	1.7	1.3	54
BundleFusion	**1.6**	-	-	-	1.1	2.2	**1.2**	-
DVO SLAM	2.1	4.6	5.3	1.7	-	3.5	-	-
Elastic Fusion	2.0	4.8	6.8	7.1	1.1	1.7	1.6	-
Kintinuous	3.7	7.1	7.5	3.4	2.9	3.0	3.1	-
ORB-SLAM2	**1.6**	**2.2**	**4.7**	**0.9**	**0.4**	**1.0**	1.9	-
RGBiD-SLAM	3.2	6.6	8.7	7.5	-	6.4	-	-
RGBDSLAMv2	2.6	-	8.7	5.7	-	-	-	-

Table 4.7 ATE (cm) results for the EuRoC sequences in relation to the odometry approach

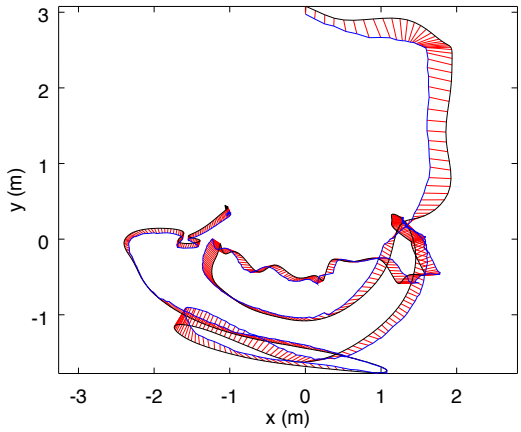
RTAB-Map	EuRoC V1			EuRoC V2			EuRoC MH					o_{avg}
Odometry	01	02	03	01	02	03	01	02	03	04	05	(msec)
F2F	8.4	6.9	x	17	89	×	3.1	4.2	12	12	10.1	40
F2M	7.1	4.0	9.7	8.2	12	×	**1.7**	2.5	6.8	16	7.6	71
Fovis	10	26	×	44	80	×	4.4	10	9.9	32	36	16
ORB2-RTAB	7.8	2.4	18	11	5.5	×	1.8	**1.5**	**2.6**	11	**5.3**	100
Viso2	11	7.3	20	13	45	×	7.8	6.3	23	25	23	80
OKVIS (IMU+stereo)	4.2	3.2	7.2	16.2	11.7	**14.4**	4.0	3.3	7.6	10.0	10.2	272
MSCKF (IMU+stereo)	6.0	4.8	13	13.5	11.9	15.7	8.8	8.7	9	16	12	**9**
LSD-SLAM (stereo)	6.6	7.4	8.9	-	-	-	-	-	-	-	-	-
ORB-SLAM2 (stereo)	**3.5**	**2.0**	**4.8**	**3.7**	**3.5**	×	3.5	1.8	2.8	12	6.0	-
SOFT-SLAM (stereo)	4.2	3.4	5.7	7.2	6.9	17.3	2.8	4.2	3.8	**9.6**	5.8	-

approaches, but is the second most computationally expensive approach. OKVIS and MSCKF are the only approaches able to track the whole V2-03-difficult sequence. Other approaches fail in this sequence when there is fast motion with a lot of motion blur, making difficult to track features: a visual inertial odometry approach is more robust to these kind of events. Fovis, F2F and MSCKF are the only real-time approaches (under 50 msec). For OKVIS and MSCKF, the processing time (on a single CPU core) is the average of image updates, excluding IMU updates that are done under 1 ms. For V1 and V2 sequences, ORB2-RTAB performs worst than the results reported in the ORB-SLAM2 paper [Mur-Artal et Tardós, 2017]: global bundle adjustment performed by ORB-SLAM2 on loop closures would then give better optimization than only using graph optimization done by RTAB-Map for these sequences. The opposite is observed however for the MH

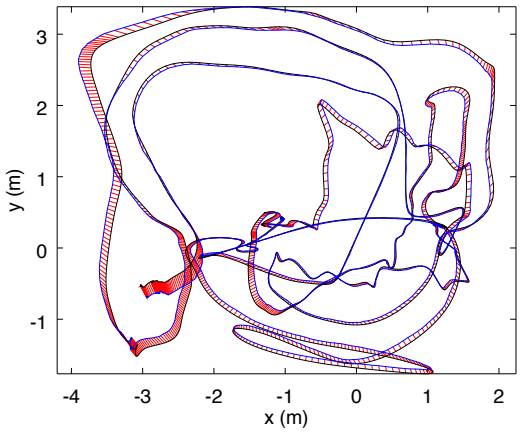


(a) V1-02-medium, F2M

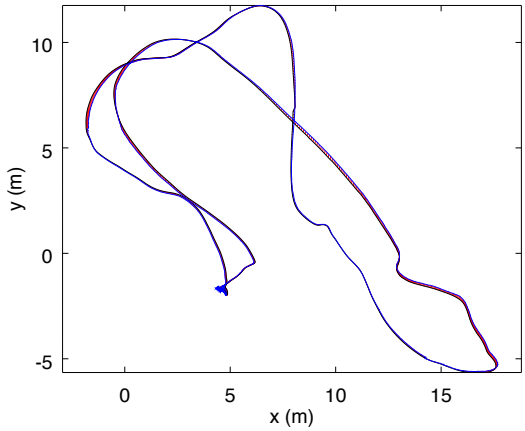
(b) V1-02-medium, OKVIS



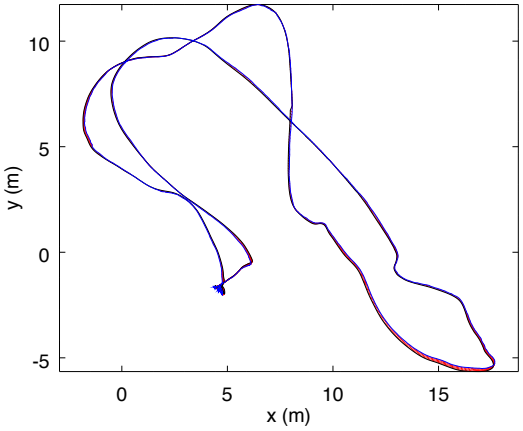
(c) V2-03-difficult, F2M



(d) V2-03-difficult, OKVIS



(e) MH-04-difficult, F2M



(f) MH-04-difficult, OKVIS

Figure 4.11 Trajectories using RTAB-Map with stereo odometry F2M (left) and visual-inertial odometry OKVIS (right) against ground truths for three EuRoC sequences. Errors between poses estimated by RTAB-Map (blue) and the ground truths (black) are shown in red.

sequences. LSD-SLAM has only been tested on V1 sequence, and results are slightly better than F2M on two out of three sequences.

4.4.4 MIT Stata Center

MIT Stata Center dataset is a collection of ROS bags recorded on a PR2 robot teleoperated in an office environment. The two sequences we use are 2012-01-25-12-14-25 and 2012-01-25-12-33-29. These sequences were chosen because they have many sensors: 2D lidar data, stereo images, RGB-D images and combined wheel and IMU odometry, making them perfect to compare different configurations of RTAB-Map. When replaying the ROS bags, the sensor data are published at the same rate as on the robot, allowing to test online capabilities of SLAM algorithms. Also, these two sequences overlap, allowing to test multi-session SLAM. The laser scans recorded in these bags are from a 2D long-range lidar of 30 m (UTM30) at 40 Hz. To test with a short-range lidar, ROS *laser_filters* package [17] was used to filter the scans up to a maximum distance of 5.6 m to emulate a lower cost short-range lidar (like an URG04LX). For S2M and S2S approaches using the lidar, laser scans are downsampled using a voxel filter of 5 cm, then normals are computed during the Point Cloud Filtering step. Point to plane ICP registration is done with `Icp/PointToPlaneMinComplexity` set to 0.02.

To make comparison possible using the MIT Stata Center dataset, the following issues had to be addressed:

– Because these ROS bags are large (30 to 50 GB) and a lot of data have to be streamed in real-time, the hard-drive on the computer had some difficulties to correctly replay the bags, sometimes causing lags and message dropping. This is particularly annoying with visual odometry approaches for which a constant stream of images should be received to avoid getting lost; lidar approaches are less affected because the large field of view of the laser scans let them recover from almost any orientation. For this reason, a stereo and a RGB-D bags were created for each sequence with images at 15 Hz instead of 30 Hz, allowing the computer to stream images without lagging. Visual odometry approaches are tested using these 15 Hz bags, and lidar approaches are tested using the original bags. For visual odometry approaches able to process images faster than 15 Hz on the target computer, this could indeed impact negatively their performance in comparison than using images at 30 Hz. However, based on our experiments, 15 Hz is a good trade-off between able to process all images online

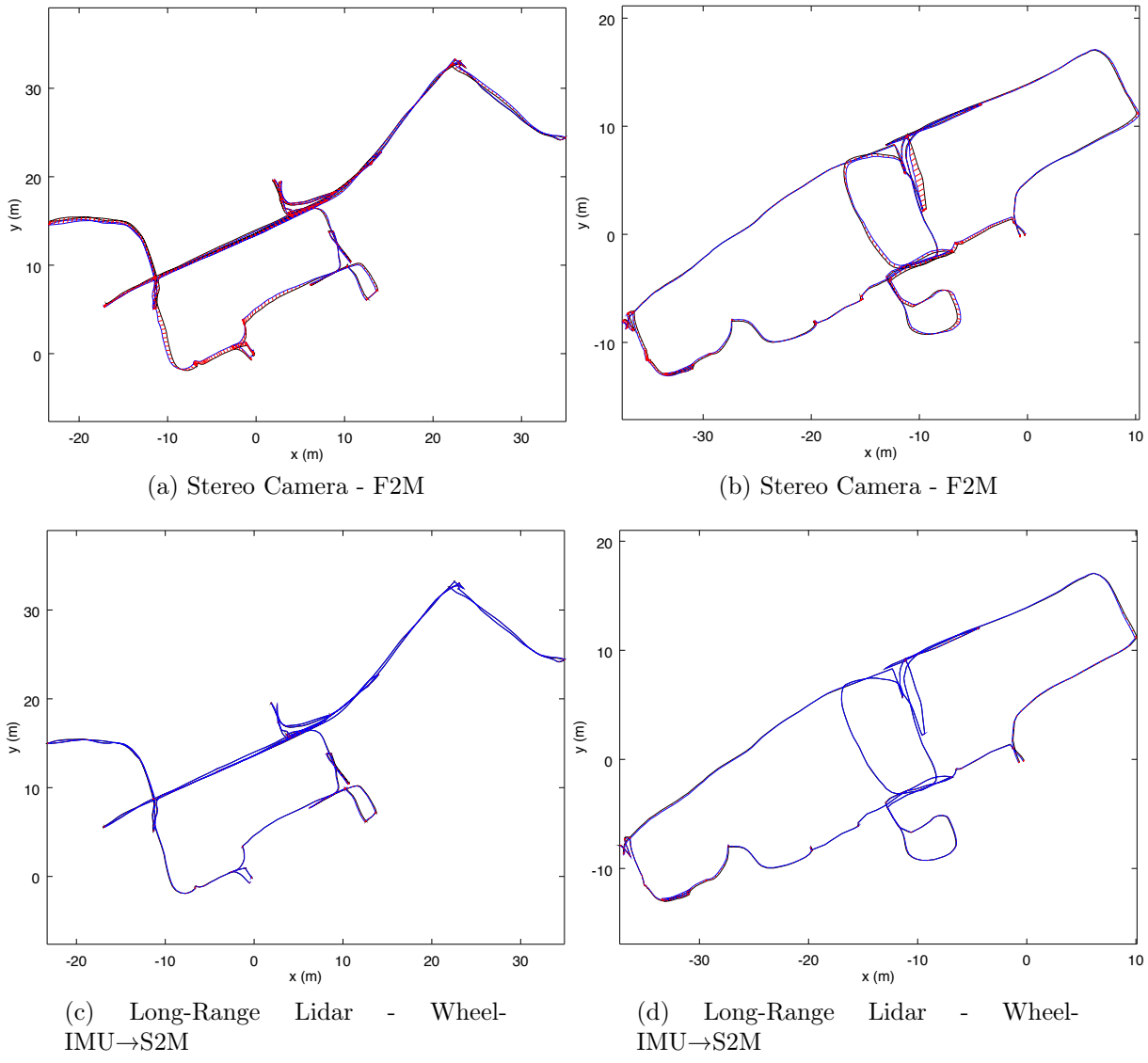
17. `http://wiki.ros.org/laser_filters`

while not getting lost (at the speed of the PR2), thus getting a better comparison of their performance independently of the frame rate.

- We observed that the scale of stereo or RGB-D data are slightly off in comparison to lidar data: a factor of 1.091664 has been applied to stereo baseline for stereo camera setup, and a factor of 1.043 has been applied to scale the depth image for RGB-D camera setup.
- The lidar is located on the base and the cameras on the head of the robot, looking directly forward. The ground truth included in the bag refers to /laser_link frame. Therefore, a special node has been then created to transform it back in /base_footprint frame so that all approaches (e.g., lidar-based or visual-based) refer to same base frame on the robot.
- We also observed that there is an offset with the timestamps between the ground truth frame and the topics, which is probably a technical error caused when the ground truth was recorded in the rosbag. Therefore, republishing the ground truth in /base_footprint frame is done with offset of 82.2 sec to synchronize it with the other topics.

ATE values are computed at each frame to see their evolution over time. ATE_{max} is the maximum error during the experiment, and ATE_{end} is the error at the end of the experiment. ATE_{max} is a indication of which approach is better for autonomous navigation minimizing odometry drift, and ATE_{end} is a indication about how well the final map represents the environment. As the robot is moving relatively slow and sequences are long, `Rtabmap/DetectionRate` is set to 1 Hz to minimize memory usage, and `Mem/STMSize` is set to half the size (15). WheelIMU is a new odometry type introduced in comparison with experiments done with the other datasets: WheelIMU is the odometry computed by combining odometry estimated by wheel encoders and the IMU using an Extended Kalman Filter [Marder-Eppstein *et coll.*, 2010], which is already available in the bags. When WheelIMU is set as prefix to S2M and S2S approaches, it means that WheelIMU is fed as external odometry estimation to S2M or S2S. $\text{WheelIMU}_{\text{refined}}$ indicates that neighbor links are refined using the laser scans when a new node is added to STM in the mapping module (`RGBD/NeighborLinkRefining` is true). For lidar-based odometry approaches, the RGB-D camera is used for loop closure detection in RTAB-Map. Stereo camera could also be used for loop closure, but computing motion estimation with RGB-D images is slightly faster than with stereo images (avoiding stereo correspondences computation).

Figure 4.12 illustrates trajectory comparison between a stereo-based approach (F2M) and a long-range lidar-based approach (WheelIMU→S2M). While the final results are roughly



(a) Stereo Camera - F2M

(b) Stereo Camera - F2M

(c) Long-Range Lidar - Wheel-IMU→S2M

(d) Long-Range Lidar - Wheel-IMU→S2M

Figure 4.12 Trajectories using RTAB-Map (blue) against ground truths (black) for the 2012-01-25-12-14-25 (left) and 2012-01-25-12-33-29 (right) Stata Center sequences using stereo camera (top) or long-range lidar (bottom). Errors between poses estimated by RTAB-Map and the ground truths are shown in red.

similar, the lidar-based approach follows the ground truth almost perfectly. Table 4.8 presents the resulting ATE performance for each sequence. Long-range lidar configurations are the most accurate (lowest ATE_{end} and ATE_{max}), and there are not so much differences between S2M and S2S approaches using WheelIMU or not. For short-range lidar, it is better to use WheelIMU→S2M over WheelIMU→S2S. The poor results of S2M and S2S with short-range lidar are caused by the corridors in the sequences: as explained in Section 4.3.4, when entering a corridor with a constant speed and a short-range lidar, the robot

Table 4.8 Online results for the MIT Stata Center 2012-01-25-xx-xx-xx sequences in relation to the sensor used and the odometry approach

Sensor	Odometry	12-14-25 ATE_{end} (m)	12-14-25 ATE_{max} (m)	12-33-29 ATE_{end} (m)	12-33-29 ATE_{max} (m)	o_{avg} (msec)
Long-Range Lidar	WheelIMU→S2S	0.06	0.08	0.08	**0.09**	**15**
	WheelIMU→S2M	**0.05**	**0.05**	0.08	**0.09**	25
	S2S	**0.05**	0.08	**0.07**	0.10	**15**
	S2M	**0.05**	0.06	**0.07**	0.10	25
	$WheelIMU_{refined}$	0.07	0.10	0.11	0.11	-
	WheelIMU	0.09	0.26	0.10	1.13	-
Short-Range Lidar	WheelIMU→S2S	0.26	0.27	0.63	0.70	**15**
	WheelIMU→S2M	**0.07**	**0.08**	**0.09**	**0.10**	22
	S2S	11	11	4.47	4.47	**15**
	S2M	4.61	4.64	1.27	1.28	22
	$WheelIMU_{refined}$	**0.07**	0.12	0.11	0.14	-
	WheelIMU	0.12	0.39	0.12	2.23	-
Stereo Camera	F2M	0.30	0.47	0.23	**0.47**	60
	F2F	0.28	0.61	0.34	1.09	**40**
	Fovis	×	×	×	×	×
	ORB2-RTAB	0.227	**0.31**	0.37	0.57	45
	Viso2	0.88	3.64	0.71	1.4	100
	WheelIMU	**0.12**	1.10	**0.16**	3.95	-
RGB-D Camera	F2M	0.37	0.91	0.38	**0.80**	54
	F2F	0.38	0.69	0.42	0.94	**32**
	Fovis	×	×	×	×	×
	ORB2-RTAB	0.28	**0.64**	0.49	0.85	44
	DVO	0.56	0.77	0.55	1.62	45
	WheelIMU	**0.11**	1.30	**0.19**	3.60	-

cannot know if it is accelerating or decelerating while seeing only two parallel lines, so with the constant motion assumption, it drifts along the corridor direction until it reaches the end of the corridor; for WheelIMU→S2M and WheelIMU→S2S approaches, this does not happen as the external odometry (using the wheel and IMU combined) can reveal that the robot is stopping at the middle of the corridor for example.

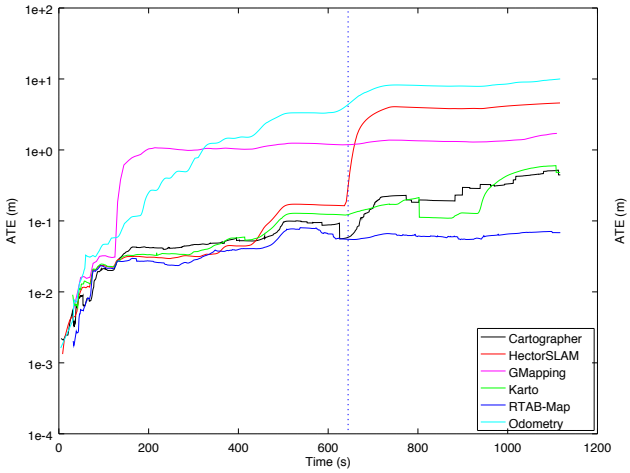
In term of computation time, lidar odometry approaches are faster than visual odometry ones (lowest o_{avg}), with S2S approaches faster for all lidar experiments. Note that WheelIMU approaches do not have any computational cost (o_{avg}) because this is the odometry saved in ROS bags used directly as input to RTAB-Map. Since accuracy is similar, one may be tempted to choose $WheelIMU_{refined}$ over WheelIMU→S2M to minimize computa-

tion cost, but the advantage of the later is for navigation: the lidar odometry can be used as odometry input for other ROS modules which should run independently of the mapping module at higher frame rate (e.g., move_base's local costmap can be more accurately updated using odometry from WheelIMU→S2M than WheelIMU).

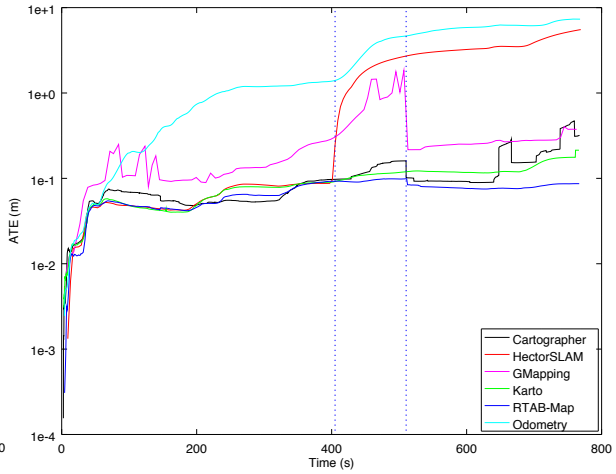
Therefore, lidar SLAM outperforms visual SLAM in this kind of environment. Comparing only visual approaches, stereo input gives better results than RGB-D input. This can be explained by the poor depth accuracy of the RGB-D sensor at range greater than 4 meters. In contrast, with a stereo camera, farther features have better depth estimation, which helps improve motion estimation. ORB2-RTAB and F2M perform better on the first and second sequences, respectively. With input images at 15 Hz, all visual odometry approaches are able to process frames in real-time (under 66 ms). Because of its memory leak explained in Section 4.3.1, ORB2-RTAB requires a lot more RAM with 1600 MB instead of 230 MB for other approaches. As Fovis gets lost very often, it is not able to complete the trajectories. WheelIMU performs better than visual odometry approaches if only the final error ATE_{end} of the map is considered. This is mainly explained by the lack of visual features in some areas, where visual odometry can drift a lot more than WheelIMU which results in less consistent motion estimations, influencing the quality of the graph optimization. However, during mapping, WheelIMU reaches larger errors, which makes it less suitable for navigation.

Lidar-Based SLAM Comparison

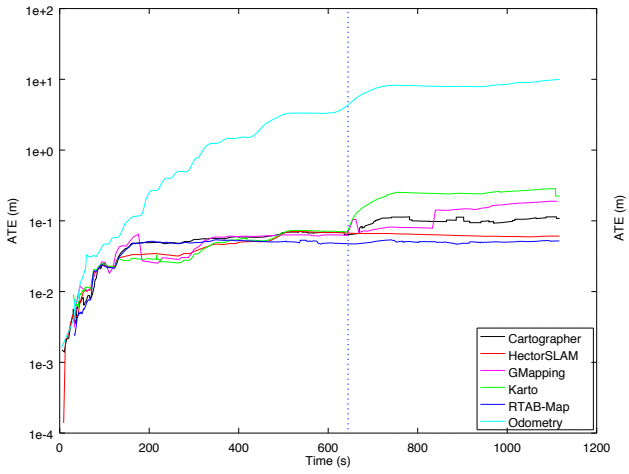
To illustrate further the capabilities of the extended version of RTAB-Map, we conducted experiments comparing RTAB-Map lidar-based SLAM using the WheelIMU→S2M configuration against other popular open source lidar-based SLAM approaches, i.e., Google Cartographer [Hess *et coll.*, 2016], Karto SLAM [Vincent *et coll.*, 2010], Hector SLAM [Kohlbrecher *et coll.*, 2011] and GMapping [Grisetti *et coll.*, 2007b]. Their ROS implementations have been used with default parameters for long-range and short-range lidar data. For Google Cartographer, GMapping and Karto SLAM, combined WheelIMU odometry is also used as input. Figure 4.13 and Table 4.9 summarize ATE results. For GMapping, as it is estimating multiple paths using its particle filter, ATE values are computed against the current best path published. An ATE of 1 m means that if at that moment the robot has to return to a specific location in the map, there is at least 1 m of error (without considering the accumulating error afterward if there is no localization) to reach it. Therefore, for autonomous navigation, ATE should be always as low as possible, and when ATE increases, it means that position estimated by the robot is drifting. If this value drifts too much, autonomous navigation to return to an known area may become impossible. To



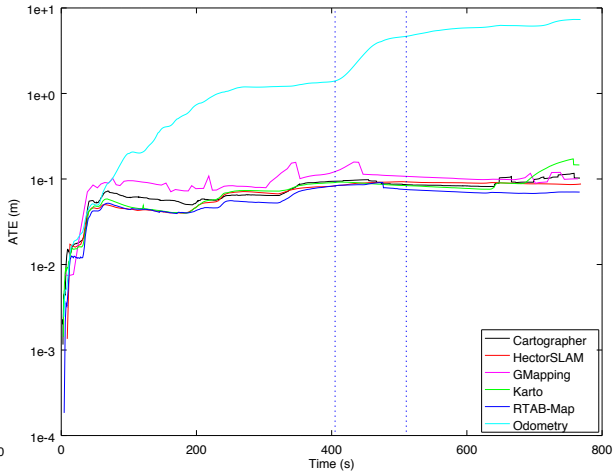
(a) 2012-01-25-12-14-25 Short-Range Lidar



(b) 2012-01-25-12-33-29 Short-Range Lidar



(c) 2012-01-25-12-14-25 Long-Range Lidar



(d) 2012-01-25-12-33-29 Long-Range Lidar

Figure 4.13 Comparison of RTAB-Map's WheelIMU→S2M with other lidar-based SLAM approaches

reduce the error, the robot has to localize itself on the map, which can be done by loop closure detection. The occurrence of loop closure detection can be observed when ATE decreases. For example, in Figure 4.13b, a large loop closure is found around 500 sec (on the second vertical dotted line). For long-range lidar, RTAB-Map's WheelIMU→S2M is the approach drifting the less, and is equal to Hector SLAM regarding ATE_{max} on the second sequence. For short-range lidar, RTAB-Map's WheelIMU→S2M is the best for both sequences. Other approaches drifted a lot in corridor sections when using short-range lidar. In particular, Hector SLAM being the only approach not using external odometry

to help scan matching, it diverges a lot more when traversing the corridors (identified by vertical lines just after 600 and 400 sec in each sequence, respectively).

4.5 Evaluating Computation Performance between Visual and Lidar SLAM Configurations with RTAB-Map

As seen in Section 4.3.3, depending on the sensor and configurations chosen, some approaches are available to create local occupancy grids and to assemble a global occupancy grid described in Section 4.3.6. The choices have an impact on computation time, memory usage and map quality. Table 4.10 presents all the configurations possible. For the local occupancy grid, the possibilities derived from Figure 4.7 involve GFD (Grid/FromDepth), G3D (Grid/3D) and GRT (Grid/RayTracing). The global occupancy grid presented in Figure 4.8 involves 2D (2D Global Occupancy Grid), OctoMap 2D (2D Global Occupancy Grid from OctoMap projection) and OctoMap 3D (3D Global Occupancy Grid). Results are presented using the 2012-01-25-12-14-25 sequence of the MIT Stata Center dataset, with `Grid/CellSize` set to 5 cm and `Grid/MaxGroundAngle` set to 45°. Time for the local occupancy grid refers to the time required by STM to create the local occupancy grid. The times for the global occupancy grid is the time needed to update the global occupancy grid: the Update time is the time required to assemble the new local occupancy grid to the global occupancy grid; the Pub time is the time required to serialize the global occupancy grid and to publish it as a ROS topic; and the With Loop time is the additional time required to re-assemble the whole global occupancy grid when the graph has been optimized after a loop closure. As expected, generating 2D local occupancy grids from 2D lidar data is faster than from depth or disparity images, as there are less points to process. When using stereo input, an additional 10 msec is required to compute the disparity image in comparison to RGB-D, for which the depth image can be used directly. Ray tracing in 2D for camera inputs adds 1 msec (e.g., 14 msec vs 13 msec for RGB-D camera).

Figure 4.14 presents examples of generation of local 2D occupancy grids depending on the sensor and approach used. Lidar-based grid provides a larger field of view, but only obstacles at the height of the lidar can be detected. In comparison, RGB-D based grid can detect some obstacles that lidar cannot, like the yellow chair, making navigation safer in this kind of environment. However, as shown in the top left of Figure 4.14g, obstacles detected by RGB-D camera after 5 m lack accuracy. Stereo camera can detect most of the obstacles as long as they are textured. It has a larger field of view than a RGB-D camera,

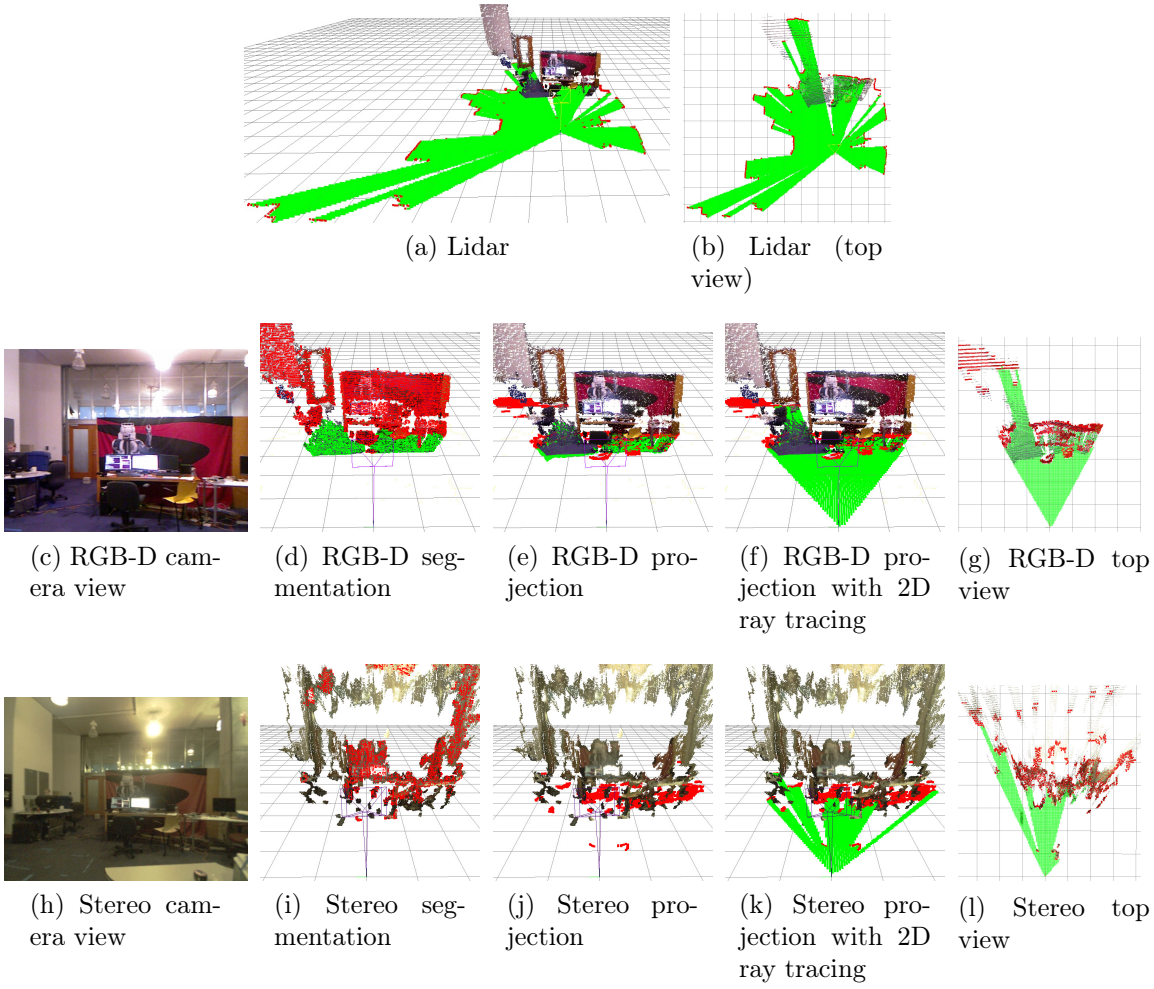
Table 4.9 ATE (m) results of RTAB-Map's WheelIMU→S2M and popular lidar-based SLAM approaches on 2012-01-25 sequences

Sensor	Odometry	SLAM	12-14-25		12-33-29	
			ATE_{end}	ATE_{max}	ATE_{end}	ATE_{max}
Long-Range Lidar	WheelIMU→S2M	RTAB-Map	**0.05**	**0.05**	**0.08**	**0.09**
	WheelIMU	Cartographer	0.11	0.11	0.10	0.12
	WheelIMU	GMapping	0.19	0.19	0.10	0.16
	WheelIMU	Karto SLAM	0.22	0.29	0.15	0.17
	-	Hector SLAM	0.06	0.07	0.09	**0.09**
Short-Range Lidar	WheelIMU→S2M	RTAB-Map	**0.07**	**0.08**	**0.09**	**0.10**
	WheelIMU	Cartographer	0.45	0.52	0.32	0.47
	WheelIMU	GMapping	1.71	1.71	0.38	1.84
	WheelIMU	Karto SLAM	0.48	0.60	0.21	0.21
	-	Hector SLAM	4.59	4.59	5.53	5.53

Table 4.10 Occupancy grid performance using MIT Stata Center 2012-01-25-12-14-25 sequence after 860 nodes added to the graph (or 350 meters in 19 minutes)

Local Occupancy Grid			Global Occupancy Grid		
Type	Sensor	Time (msec)	Type	Time Update+Pub (msec)	Time With Loop (msec)
GFD_0→2D	Long-Range Lidar	4	2D	2+0	+600
GFD_0→2D	Short-Range Lidar	1	2D	1+0	+200
GFD_1→$G3D_0$	RGB-D Camera	13	2D	1+0	+40
→GRT_0→2D	Stereo Camera	23	2D	1+0	+25
GFD_1→$G3D_0$	RGB-D Camera	14	2D	1+0	+90
→GRT_1→2D	Stereo Camera	24	2D	1+0	+90
GFD_1→$G3D_1$	RGB-D Camera	13	OctoMap 3D	2+100	+1600
→GRT_0→3D	Stereo Camera	23	OctoMap 3D	2+80	+1100
GFD_1→$G3D_1$ →GRT_1 →3D	RGB-D Camera	120	OctoMap 2D	15+540	+11300
	Stereo Camera	96	OctoMap 2D	20+670	+14800
	RGB-D Camera	120	OctoMap 3D	15+430	+13500
	Stereo Camera	96	OctoMap 3D	20+640	+15500

detecting the table just in front of the robot while the RGB-D camera cannot see it. Note that with this 2D ray tracing approach and because the cameras are not close to ground, if the robot is approaching the table from the front so that the cameras do not see it anymore, the obstacles previously added to global occupancy grid map because of the table will be incorrectly cleared. To use 2D ray tracing, it is preferred to lower the camera at the height of the smallest obstacle in the environment. If lowering the camera is not possible because of some robot physical constraints, using 2D local occupancy grids without ray tracing would then be safer, and dynamic obstacles would only be cleared if the camera can see



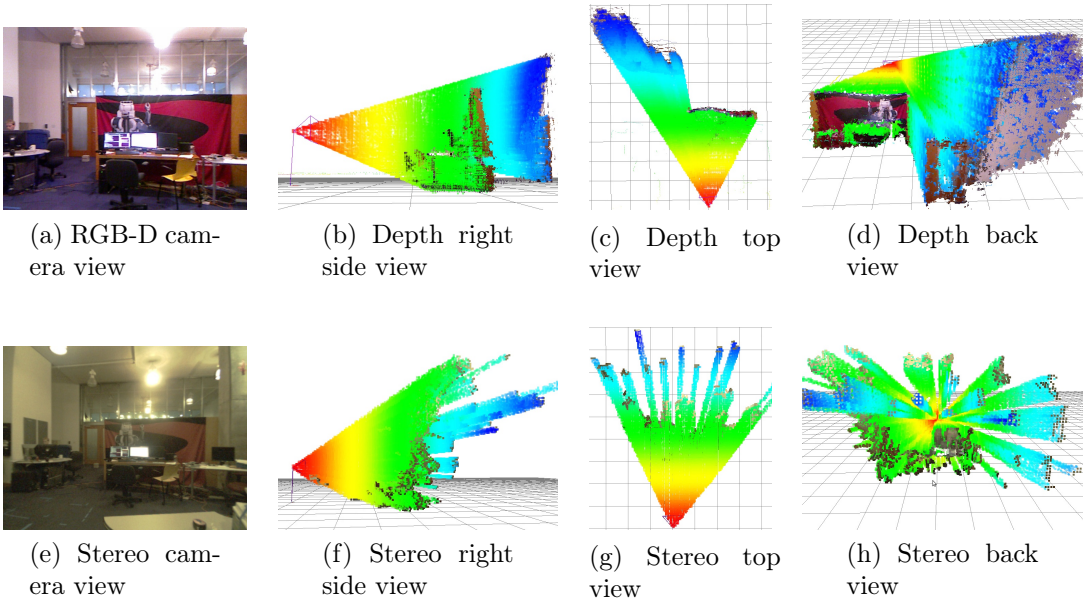
(a) Lidar (b) Lidar (top view)

(c) RGB-D camera view (d) RGB-D segmentation (e) RGB-D projection (f) RGB-D projection with 2D ray tracing (g) RGB-D top view

(h) Stereo camera view (i) Stereo segmentation (j) Stereo projection (k) Stereo projection with 2D ray tracing (l) Stereo top view

Figure 4.14 Local occupancy grid examples. For cameras, segmentation examples correspond to 3D local occupancy grids without ray tracing, then other ones are 2D local occupancy grids without or with 2D ray tracing. Obstacle cells are shown in red. Empty and ground cells are shown in green. The black grid is only a visual reference and has cell size of 1 m.

the ground where the obstacle was (as in Figure 4.14d). Note that stereo camera cannot see the ground (see Figure 4.14i), so dynamic obstacles cannot be cleared without ray tracing. Another solution to solve the problem of safe obstacle clearing is to use 3D local occupancy grids with ray tracing and OctoMap, at the cost of significantly increasing the processing power required. Figure 4.15 illustrates examples of 3D ray tracing with RGB-D and stereo cameras. As depth images are more dense than disparity images, ray tracing takes more time to do using RGB-D camera. However, as stereo cameras have a larger field of view, more volume would be filled, creating larger local occupancy grids. This explains why updating 3D global local occupancy grids from stereo camera takes more time than from RGB-D cameras. If a 3D occupancy grid is required and the environment is static



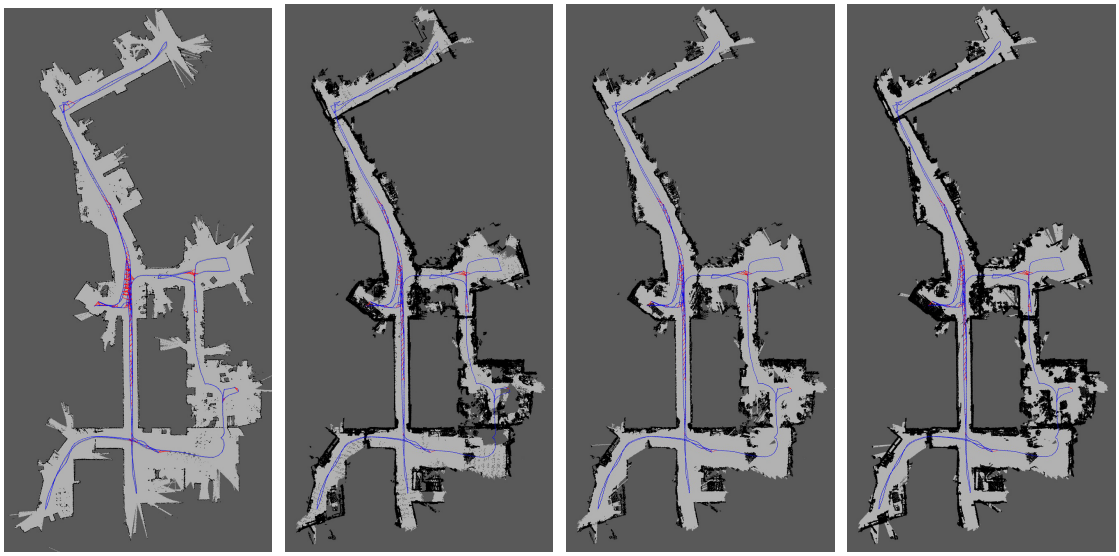
(a) RGB-D camera view (b) Depth right side view (c) Depth top view (d) Depth back view

(e) Stereo camera view (f) Stereo right side view (g) Stereo top view (h) Stereo back view

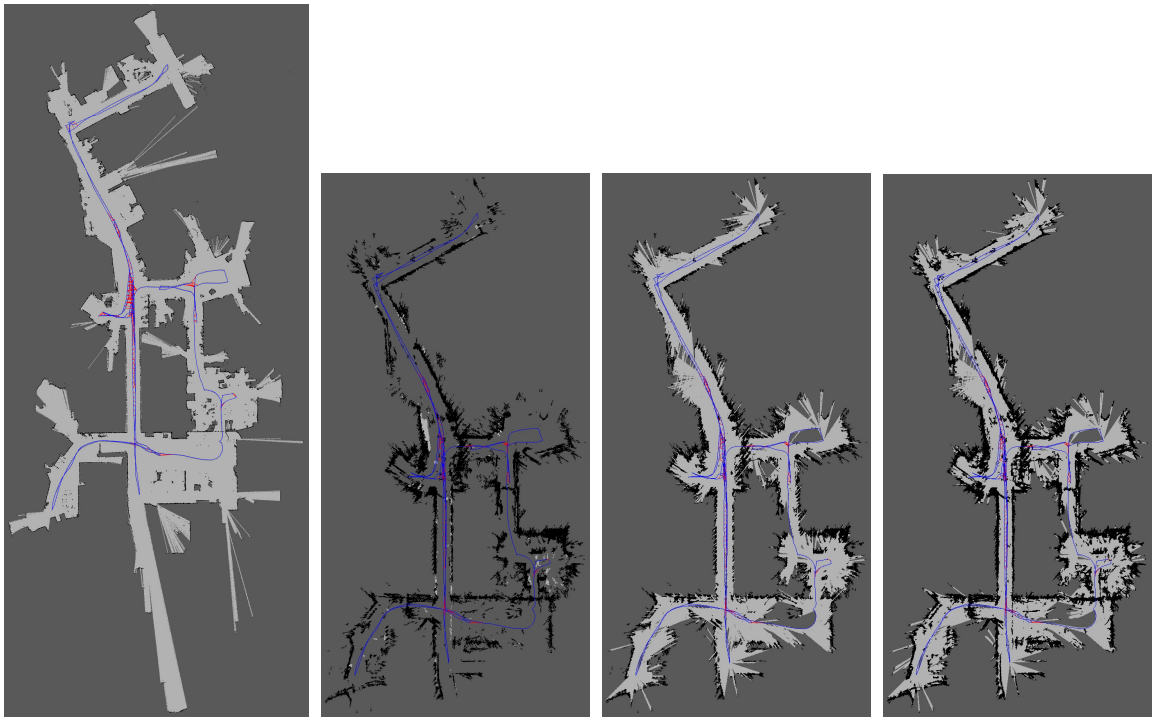
Figure 4.15 3D local occupancy grid map with ray tracing examples

(no dynamic obstacles to clear), avoiding ray tracing can help save a lot of computing resources.

Figure 4.16 presents the corresponding 2D global occupancy grids created for the different sensors and local occupancy grid approaches of Table 4.10. Lidar-based maps give the most accurate geometry of the environment (at the height of the lidar), followed by maps created from a RGB-D camera. Without ray tracing, stereo-based map contains almost no empty cells because the disparity approach cannot see texture-less ground. Some walls are also not detected or very noisy. Using ray tracing, empty space can be filled. When comparing 2D ray tracing against 3D ray tracing, it is possible to see that some obstacles were incorrectly cleared using 2D ray tracing: beside the doors, the environment is considered static so obstacles should not have been cleared. For 3D ray tracing, when opening the door, if the field of view of the camera cannot see the whole opening (e.g., camera cannot see the bottom of the door), it will be able to clear only the volume of the door it can see, leaving the bottom as obstacle. Figure 4.17 presents the 3D occupancy grid of the OctoMap at tree depth 16 and 14 using the RGB-D camera. Tree depth 16 corresponds to cell size of 5 cm and shown with RGB color. Generating the OctoMap at lower tree depth increases cell size (lower resolution), which can be useful for faster path planning.

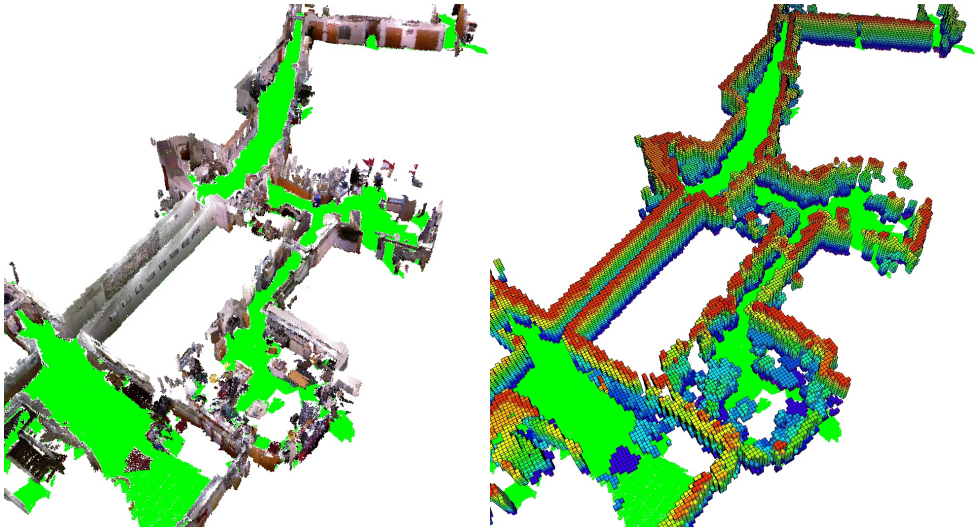


(a) Short-range lidar (b) RGB-D projection without ray tracing (c) RGB-D projection with 2D ray tracing (d) RGB-D OctoMap projection with 3D ray tracing



(e) Long-range lidar (f) Stereo projection without ray tracing (g) Stereo projection with 2D ray tracing (h) Stereo OctoMap projection with 3D ray tracing

Figure 4.16 2D occupancy grid map examples.



(a) (b)

Figure 4.17 OctoMap of depth a) 16 and b) 14 using the RGB-D camera.

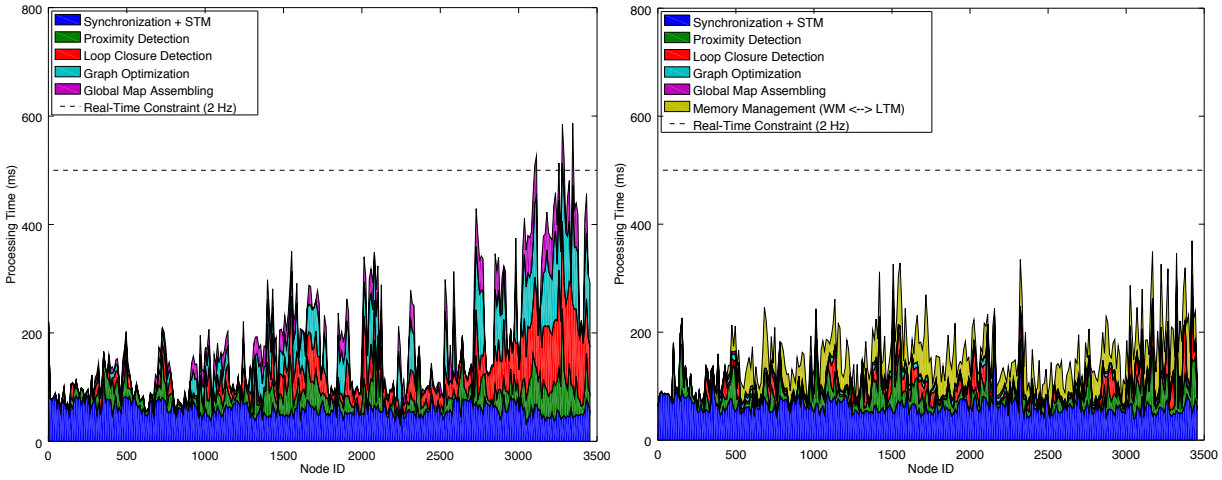
4.5.1 Examining the Use of RTAB-Map's Memory Management Mechanism

For large-scale and long-term SLAM where the graph is constantly adding new nodes, these previous solutions to adjust computation load based on occupancy grid type may not be sufficient. In all previously described experiments, RTAB-Map's memory management mechanism was disabled to always have access to global map for trajectory accuracy and occupancy grid comparisons. To outline how much time is required for each module of RTAB-Map's WM in Figure 4.1 in a large scale environment, the two MIT Stata Center sequences were played back to back, creating a long mapping experiment containing two mapping sessions linked together. Both sequences start and finish at the same location, so a loop closure between the end of the first sequence and the beginning of the second sequence can be detected when playing the second bag, merging automatically the two maps. As the ground truth coordinates between the two bags are slightly off, the ground truth of the second bag is transformed in the same coordinates than the ground truth of the first bag. To do so, we assembled the scans for each sequence separately using their respective ground truths, then using *pcl_icp* tool[18], the two point clouds are registered and the resulting transformation $(x, y, \theta) = (0.006236 \text{ m}, -0.351500 \text{ m}, -0.017832 \text{ rad})$ can be applied to the ground truth of the second bag. RTAB-Map's update rate `Rtabmap/DetectionRate` is also increased to 2 Hz to add twice the nodes to the graph, with `Mem/STMSize` set back to 30 so that nodes stay the same time in STM than at 1 Hz. WM is limited to a maximum

18. `https://github.com/PointCloudLibrary/pcl/blob/master/tools/icp.cpp`

size `Rtabmap/MemoryThr` of 300 nodes to better observe the effect of memory management when a low number of nodes is kept in WM. RTAB-Map's odometry configuration used is WheelIMU→S2M with short-range lidar. From Table 4.8, while accuracy results are slightly worst than with long-range version, using short-range lidar requires significantly less time to regenerate the global occupancy grid as shown in Table 4.10 for a similar quality (Figure 4.16a versus Figure 4.16e).

Figure 4.18 presents the timing results without (a) and with (b) memory management for RTAB-Map WM modules presented in Figure 4.1. The horizontal line is the real-time constraint, i.e., the maximum time allowed for addition of new nodes to map at 2 Hz. In that case, RTAB-Map's WheelIMU→S2M without memory management is not satisfying real-time constraints, because some updates require more than 500 msec. Between nodes 3000 and 3500, the robot is revisiting an area which has been already visited multiple times (e.g., the beginning and ending areas of each sequence), triggering many more loop closure and proximity detections with previously mapped paths. As the map increases in size, the update time increases, which creates large holes in the map if there are no new nodes added during a number of seconds if the robot is moving. With memory management, RTAB-Map's WheelIMU→S2M is able to satisfy real-time constraints for the whole experiment. Memory management adds a small overhead (average of 52 msec) when moving nodes between WM and LTM, but it greatly reduces the processing time required for other modules depending on graph size. However, the global occupancy grid map does not always represent the full environment visited. Figure 4.19a shows five global occupancy grid examples at different time in the experiment using memory management. The blue triangle indicates the robot pose at the referred time. At $t = 225$ sec, $t = 1170$ sec and $t = 1400$ sec, the robot is moving to a new area. At $t = 460$ sec and $t = 930$ sec, the robot is revisiting a previously mapped area, explaining why there is an area of the map in front of the robot. When revisiting a previously visited area, memory management retrieves nodes from LTM to WM to expand the current map with old locations. While the online global occupancy grid map is limited around the current pose of the robot, using its memory management mechanism, RTAB-Map is still able to detect most of the loop closures needed to correctly merge the two sessions. Without memory management, there are 2048 loop closure links and 1129 proximity links, in comparison to 1256 loop closure links and 774 proximity links with memory management. For instance, Figure 4.19b and Figure 4.19c are the global occupancy grid maps created from the experiments with and without memory management, respectively. The map in (b) is generated after online mapping using all links saved in LTM. Both experiments give the same final ATE of 12 cm.



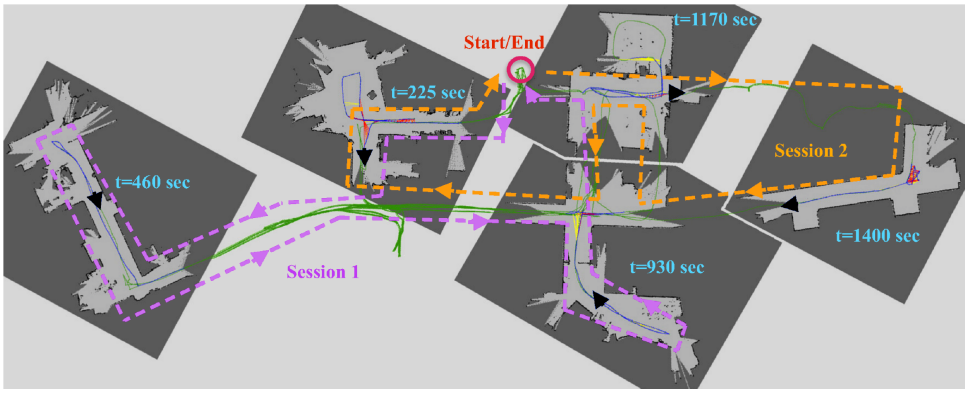
(a) (b)

Figure 4.18 Processing time required for each module inside *rtabmap* ROS node without (a) and with (b) memory management, for a map update rate of 2 Hz using the combined sessions of the MIT Stata Center sequences.

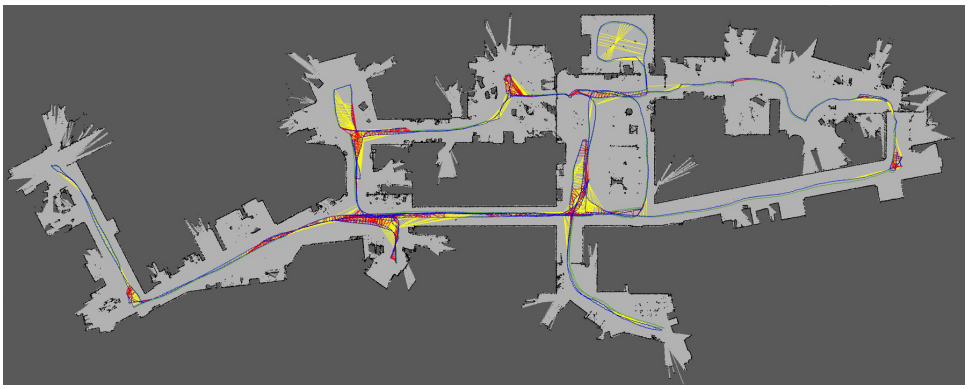
4.6 Discussion

Trajectory performance evaluation presented in Section 4.4 demonstrates that integrating odometry approaches in our extended version of RTAB-Map can lead to results comparable to state-of-the-art visual-based and lidar-based SLAM approaches, making it a powerful library for the design and prototyping SLAM with different sensors. To our knowledge, using our extended version of RTAB-Map, this paper is the first to report such experimental comparison of lidar versus visual-based SLAM configurations on the same system. While being also ROS ready for 2D and 3D online autonomous navigation, this makes the approach easy to integrate to a custom robot in order to compare live the differences between visual and lidar SLAM configurations. It is often difficult to compare these SLAM configurations when they have been tested on datasets that only work with their sensor type and derived simply by teleoperating the robot or by having a human positioning the sensor. Some configurations also require that the sensor moves in a specific way to compensate for their limitations. These live comparisons can then help to reveal flaws and limitations inherent to the sensor chosen when combined with standard navigation approaches like ROS' navigation stack [Marder-Eppstein *et coll.*, 2010].

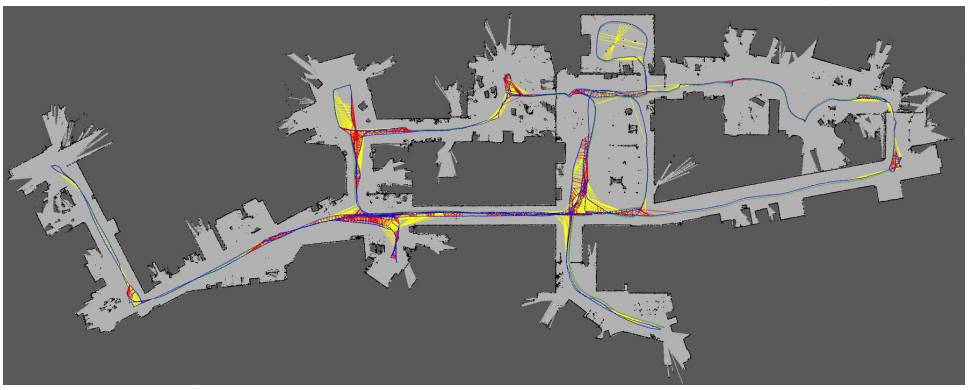
Consequently, RTAB-Map can be used to conduct trials with different sensors and identify early on if a sensor is suitable for the targeted application. Based on the results presented in this paper, guidelines can be derived regarding when using SLAM (without external global localization) in an indoor environment. Unless a long-range lidar is used, having



(a)



(b)



(c)

Figure 4.19 Global maps created with (a,b) and without (c) memory management using the combined sessions of the MIT Stata Center sequences. In (a), five examples of the biggest map created online using available nodes in WM at the specified time are presented, with the black triangle as the robot pose at that time. The global trajectories done during each session are shown in purple and orange, respectively. Neighbor, loop closure and proximity links are shown in blue, red and yellow, respectively. The green path is the ground truth, which is almost not visible in (b) and (c) as the blue line is superposed.

odometry input from proprioceptive sensors (e.g., IMU, wheel encoders) is mandatory for robust autonomous navigation. When relying only on short-range sensors, it is very likely that the robot will end in an area where the sensor cannot see enough features to be able to localize itself in its map. For cameras, seeing a white wall, a textureless or a dark area would result in loosing localization. For short-range lidar, a large empty space or a long corridor with low geometry complexity can be also problematic. Both kind of sensors have then their issues depending on the environment.

In RTAB-Map, motion estimation during localization or loop closure detection is done primarily visually, then optionally refined by geometry if a lidar is available. This means that if the visual motion estimation fails, lidar motion estimation cannot be done. In future work, tighter coupling of visual and geometry estimations could be evaluated so that if one fails, the other can still be used to get an estimation of the position. This also applies to odometry, where a visual-lidar approach could be more robust when environments are textureless or lacking geometry. For loop closure detection, the current bag-of-words approach is dependent on a camera, meaning that a camera is always required even if lidar SLAM is done. As a solution, it is possible to feed a fake empty image to RTAB-Map if the robot does not have a camera, relying only on proximity detection for map corrections. As long as the robot is not drifting too much and the environment is relatively small (e.g., a single building), proximity detection could detect most of the loop closures without needing a camera. For large-scale loop closure detection where pose estimation cannot be used robustly, a lidar-based loop closure detection could be integrated (similar to [Bosse et Zlot, 2008] and [Hess *et coll.*, 2016]) so that the robot can detect very large loop closures only using a lidar (even in completely dark environments).

Based on the results, general observations can be made regarding sensor choice for indoor navigation. While stereo cameras give slightly better localization accuracy, RGB-D cameras are preferred because textureless surfaces can be detected for obstacle avoidance. For all exteroceptive sensors based on light, navigation in environments filled with glass and reflective objects can be unsafe, as obstacles cannot be detected or false obstacles will be added to the map. In term of cost, using a RGB-D camera can be beneficial compared to a lidar. However, the extra field of view of the lidar is a huge advantage over a single RGB-D camera when it comes to low-drift navigation, as shown in Section 4.4.4 with lower ATE_{max}. One could add more RGB-D cameras to get a field of view similar to the lidar, but the increase in multi-camera calibration complexity and computational load do not justify the cost for 2D navigation robustness unless 3D obstacles must be detected, as a low-cost lidar can do the job using less computing resources. However for 3D navigation (e.g.,

drone), having a multi-camera setup is beneficial to get a larger field of view compared with expensive 3D lidars (e.g., Velodyne) in term of cost. In our use of RTAB-Map on real robots, depending of the projects and the target cost, we had to deal with the above limitations of the sensors used. One example is a patrolling robot doing continuously SLAM while autonomously navigate in the environment ([Labbé et Michaud, 2017]). The robot was equipped with a RGB-D camera, a short-range lidar and wheel odometry. The environment was mostly long textureless corridors, making it difficult to localize visually and also geometrically. Visual loop closures could only be found at the end of corridors or in the rooms. The use of wheel odometry was then mandatory. Proximity detection helped alignment with the corridor using geometry in almost any directions (the lidar had more than 180° of field of view). The lidar's large field of view also helped during navigation when the robot had to avoid an obstacle, allowing it to localize even if the robot was oriented differently during the mapping. Another example is its use on a low cost autonomous wheelchair using only a RGB-D camera for SLAM and navigation [Burhanpurkar *et coll.*, 2017]. To handle textureless corridor environments, wheel odometry was used instead of visual odometry (or visual inertial odometry) to avoid getting lost as soon as entering a corridor. The limited field of view of the front facing RGB-D camera was also a problem during navigation. If the robot did not follow a very similar path than the one done previously when mapping the environment (e.g., to avoid someone passing by), the robot could get lost as loop closures or localizations could not be detected afterward. The retrofitted wheel odometry was drifting more on this platform than the previous robot (in particular if the planner was sending many commands making often the robot turn in place), increasing the problem of relocalizing after the robot avoided an obstacle. This justifies why the ATE_{max} metric presented in this paper is important for navigation: the lower the odometry drifts, the faster the localization recovery happens after the robot changes course for some reasons and has to come back to follow the original planned path. In these application examples using short-range sensors, clearing dynamic obstacles (after they moved) would not always be possible using the current ray tracing approach if the sensor rays could not "hit" something behind where the obstacles were in order to clear the space, keeping some fake obstacles on the map that can affect planning afterward. Similar to [Barsan *et coll.*, 2018], a smarter understanding of the images or laser scans could be implemented to segment dynamic objects from the static environment.

4.7 Conclusion

This paper presents the extended version RTAB-Map, which provides a full integration with ROS to handle robot's *tf*, to synchronize RGB-D, stereo, laser scan and point cloud topics, and the ability to generate occupancy grids for all sensors. As a result, RTAB-Map is now a multi-purpose graph-based SLAM approach that can be used out-of-the-box by novice SLAM users and for prototyping on robot platforms with different sensor configurations and processing capabilities. It can be used to compare performance over datasets and to conduct online evaluations. Sensors required for SLAM, whether they are low cost or expensive, all have limitations that influence localization accuracy, map quality and computing resources. RTAB-Map's flexibility is demonstrated in this paper by making meaningful comparisons between visual and lidar-based SLAM configurations, allowing to analyze which robot sensor configuration is best for indoor autonomous navigation. RTAB-Map is distributed as an open-source library and is already available to the community. RTAB-Map is currently one of the top ROS packages actively used (over 1600 questions across its forum [19], github repositories [20] and ROS Answers [21]) by the community, for low-cost SLAM with RGB-D and stereo cameras. Our goal with RTAB-Map is to continue integrating new odometry approaches lacking proper ROS integration, to facilitate comparison of SLAM configurations for autonomous navigation of mobile robot platforms.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

19. http://official-rtab-map-forum.67519.x6.nabble.com/
20. https://github.com/introlab/rtabmap, https://github.com/introlab/rtabmap_ros
21. https://answers.ros.org

CHAPITRE 5

CONCLUSION

Globalement, cette thèse décrit l'évolution de RTAB-Map, initialement un algorithme de gestion de mémoire pour le problème spécifique de détection de fermeture de boucle à long terme et à grande échelle [Labbé et Michaud, 2013], et maintenant permettant de faire du SLAM pour la navigation autonome 'en ligne' dans des environments dynamiques et non bornés. Les résultats suggèrent que la gestion de mémoire proposée permet au robot de cartographier adéquatement et continuellement un environment sans dépasser les contraintes de traitement 'en ligne'. En gardant les endroits les plus récents, les plus visités et ceux identifiés par le planificateur de trajectoire en mémoire de travail, RTAB-Map est en mesure de localiser le robot globalement (dans la session courante et entre plusieurs sessions) tout en fournissant assez d'information à l'algorithme de suivi de trajectoire pour que la navigation dans des endroits de l'environment précédemment transférés en mémoire à long terme soit possible. De plus, les résultats quantitatifs sur la qualité de la cartographie montrent que RTAB-Map est comparable à l'état de l'art en SLAM, que ce soit visuel ou avec un télémètre laser. Avec son intégration avec les approches de planification et suivi de trajectoire inclues dans ROS, ces différentes configurations de capteurs peuvent être testées rapidement dans les environments cibles afin d'extraire sur le terrain des comparisons significatives entre l'utilisation de la vision et d'un télémètre laser pour la navigation autonome.

Les contributions de RTAB-Map vont aussi au-delà de la recherche en robotique mobile présentée dans cette thèse. RTAB-Map est distribué en logiciel libre multi-plateforme (e.g., Windows, Mac OS X, Linux) sur GitHub [22] et reçoit plus de 600 visiteurs uniques par semaine (totalisant 3500 visites et 150 clones par semaine). Lancé en 2014, le forum officiel de RTAB-Map [23] contient maintenant plus de 900 topiques, dont 1900 réponses écrites par moi, en plus des 540 questions sur GitHub [24] [25] et 250 questions sur ROS Answers [26] à propos de RTAB-Map. Les compagnies qui ont utilisés RTAB-Map proviennent entre autres des secteurs de l'immobilier (e.g., reconstruction 3D intérieur), de l'automobile (e.g.,

22. http://introlab.github.io/rtabmap
23. http://official-rtab-map-forum.67519.x6.nabble.com/
24. https://github.com/introlab/rtabmap/issues
25. https://github.com/introlab/rtabmap_ros/issues
26. https://answers.ros.org

voiture autonome et de course), de l'énergie (e.g., localisation de matières radioactives), de la réadaptation (e.g., fauteuil roulant autonome), de la sécurité et de la surveillance, de l'éducation en ligne et des services d'urgence. La version de RTAB-Map pour Google Tango [27] (un téléphone mobile avec caméra de profondeur) a été utilisée par plus de 1200 usagers pour la cartographie de maisons et bureaux, de cavernes naturelles, de mines souterraines, de tunnels abandonnés, de forêts et de sites patrimoniaux. Tous ces exemples montrent l'étendu des applications pour lesquelles RTAB-Map peut être maintenant utilisé en plus de la robotique mobile, ce qui n'aurait pas été possible sans le retour d'information et les échanges continus avec tous ces utilisateurs et contributeurs.

Pour des travaux futurs, il serait intéressant d'expérimenter de nouvelles approches de sélection des noeuds à garder en mémoire de travail pour maximiser les chances de détecter des fermetures de boucle à long terme. Un exemple pourrait être une sélection aléatoire des noeuds afin d'éviter des cas pathologiques où le robot oublierait de façon permanente des endroits. D'autres approches de gestion des poids des noeuds pourraient aussi être envisagées, comme ajouter un facteur décroissant des poids pour éviter que des noeuds ayant un poids très élevé restent indéfiniment en mémoire de travail même si ces endroits ne peuvent plus ou ne seront plus jamais visités (e.g., lorsque le robot change totalement d'environnement d'opération). D'autres améliorations pourraient aussi être faites afin d'augmenter la taille de la mémoire de travail pour la même puissance de calcul, par exemple en optimisant individuellement les modules de détection de fermeture de boucle, optimisation de de graphe (e.g., optimisation différé dans un autre processus ou optimisation localisée pour éviter d'optimiser toujours le graphe au complet) et de génération de la carte pour qu'ils puissent gérer plus de noeuds dans le même temps. À long terme, toutes ces améliorations permettraient ainsi une navigation autonome 'en ligne' encore plus robuste dans un environnement d'opération non borné.

27. `https://play.google.com/store/apps/details?id=com.introlab.rtabmap`

LISTE DES RÉFÉRENCES

Angeli, A., Filliat, D., Doncieux, S. et Meyer, J.-A. (2008). Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, volume 24, numéro 5, p. 1027–1037.

Atkinson, R. et Shiffrin, R. (1968). Human memory : A proposed system and its control processes. Dans *Psychology of Learning and Motivation : Advances in Research and Theory*, volume 2. Elsevier, p. 89–195.

Baddeley, A. (1997). *Human Memory : Theory and Practice*. Psychology Press.

Barsan, I. A., Liu, P., Pollefeys, M. et Geiger, A. (2018). Robust dense mapping for large-scale dynamic environments. Dans *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, Brisbane, Australia.

Bay, H., Ess, A., Tuytelaars, T. et Gool, L. V. (2008). Speeded Up Robust Features (SURF). *Computer Vision and Image Understanding*, volume 110, numéro 3, p. 346–359.

Besl, P. J. et McKay, N. D. (1992). Method for registration of 3-D shapes. Dans *Robotics-DL Tentative*, International Society for Optics and Photonics. p. 586–606.

Biber, P. et Duckett, T. (2005). Dynamic maps for long-term operation of mobile service robots. Dans *Robotics : Science and Systems*. p. 17–24.

Booij, O., Zivkovic, Z. et Kröse, B. (2009). Efficient data association for view based SLAM using connected dominating sets. *Robotics and Autonomous Systems*, volume 57, numéro 12, p. 1225–1234.

Bosse, M., Newman, P., Leonard, J. et Teller, S. (2004). Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework. *International Journal of Robotics Research*, volume 23, numéro 12, p. 1113–39.

Bosse, M. et Zlot, R. (2008). Map matching and data association for large-scale two-dimensional laser scan-based SLAM. *International Journal of Robotics Research*, volume 27, numéro 6, p. 667–91.

Botterill, T., Mills, S. et Green, R. (2011). Bag-of-words-driven, single-camera simultaneous localization and mapping. *J. of Field Robotics*, volume 28, numéro 2, p. 204–226.

Bradski, G. et Kaehler, A. (2008). *Learning OpenCV : Computer vision with the OpenCV library*. O'Reilly Media, Inc., Sebastopol, CA, USA.

Burhanpurkar, M., Labbé, M., Guan, C., Michaud, F. et Kelly, J. (2017). Cheap or robust ? the practical realization of self-driving wheelchair technology. Dans *Proceedings International Conference on Rehabilitation Robotics*. p. 1079–1086.

Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W. et Siegwart, R. (2016). The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, volume 35, numéro 10, p. 1157–1163.

Calonder, M., Lepetit, V., Strecha, C. et Fua, P. (2010). Brief : Binary robust independent elementary features. Dans *European Conference on Computer Vision*. p. 778–792.

Carlone, L., Aragues, R., Castellanos, J. A. et Bona, B. (2012). A linear approximation for graph-based simultaneous localization and mapping. *Robotics : Science and Systems VII*, p. 41–48.

Carrera, G., Angeli, A. et Davison, A. J. (2011). Lightweight SLAM and navigation with a multi-camera rig. Dans *European Conference on Mobile Robots*. p. 77–82.

Chen, Y., Wu, F., Wang, N., Tang, K., Cheng, M. et Chen, X. (2015). KeJia-LC : A low-cost mobile robot platform—Champion of demo challenge on benchmarking service robots at RoboCup 2015. Dans Almeida, L., Ji, J., Steinbauer, G. et Luke, S., *Robot Soccer World Cup*, volume 9513. Springer, Cham, Switzerland, p. 60–71.

Churchill, W. et Newman, P. (2012). Practice makes perfect ? Managing and leveraging visual experiences for lifelong navigation. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 4525–4532.

Churchill, W. et Newman, P. (2013). Experience-based navigation for long-term localisation. *The International Journal of Robotics Research*, volume 32, numéro 14, p. 1645–1661.

Cummins, M. et Newman, P. (2011). Appearance-only SLAM at large scale with FAB-MAP 2.0. *International Journal of Robotics Research*, volume 30, numéro 9, p. 1100–1123.

Cvišić, I., Ćesić, J., Marković, I. et Petrović, I. (2018). SOFT-SLAM : Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles. *Journal of Field Robotics*, volume 35, numéro 4, p. 578–595.

Dai, A., Nießner, M., Zollöfer, M., Izadi, S. et Theobalt, C. (2017). Bundlefusion : Real-time globally consistent 3D reconstruction using on-the-fly surface re-integration. *ACM Transactions on Graphics*, volume 36, numéro 3, p. 76a.

Della Corte, B., Bogoslavskyi, I., Stachniss, C. et Grisetti, G. (2017). A general framework for flexible multi-cue photometric point cloud registration. *arXiv preprint arXiv :1709.05945*.

Dellaert, F. (2012). *Factor Graphs and GTSAM : A Hands-On introduction* (Rapport technique GT-RIM-CP&R-2012-002). Georgia Institute of Technology.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, volume 1, numéro 1, p. 269–271.

Dubé, R., Dugas, D., Stumm, E., Nieto, J., Siegwart, R. et Cadena, C. (2016). Segmatch : Segment based loop-closure for 3D point clouds. *arXiv preprint arXiv :1609.07720*.

Dubé, R., Gawel, A., Sommer, H., Nieto, J., Siegwart, R. et Cadena, C. (2017). An online multi-robot slam system for 3D lidars. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE. p. 1004–1011.

Endres, F., Hess, J., Sturm, J., Cremers, D. et Burgard, W. (2014). 3-D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, volume 30, numéro 1, p. 177–187.

Engel, J., Schöps, T. et Cremers, D. (2014). LSD-SLAM : Large-scale direct monocular SLAM. Dans *European Conference on Computer Vision*, Springer. p. 834–849.

Engel, J., Stückler, J. et Cremers, D. (2015). Large-scale direct SLAM with stereo cameras. Dans *Proceedings IEEE/RSJ International Conference onIntelligent Robots and Systems*. p. 1935–1942.

Fallon, M., Johannsson, H., Kaess, M. et Leonard, J. J. (2013). The MIT stata center dataset. *The International Journal of Robotics Research*, volume 32, numéro 14, p. 1695–1699.

Ferland, F., Clavien, L., Frémy, J., Letourneau, D., Michaud, F. et Lauria, M. (2010). Teleoperation of AZIMUT-3, an omnidirectional non-holonomic platform with steerable wheels. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 2515–2516.

Folkesson, J. et Christensen, H. I. (2007). Closing the loop with graphical SLAM. *IEEE Transactions on Robotics*, volume 23, numéro 4, p. 731–41.

Foote, T. (2013). tf : The transform library. Dans *Proceedings IEEE International Conference on Technologies for Practical Robot Applications*. Open-Source Software workshop, p. 1–6.

Foresti, H., Finch, G., Cavalcanti, L., Alves, F., Lacerda, D., Brito, R., Verde, F. V., Barros, T., Freitas, F., Lima, L., Ribeiro, W., Mabuse, H., Teichrieb, V. et Teixeira, J. M. (2016). Emotive robotics with I-Zak. http ://www.robocup2016.org/media/symposium/Team-Description-Papers/AtHome/RoboCup_2016_AtHome_TDP_CESAR_VOXAR_LABS.pdf.

Forster, C., Pizzoli, M. et Scaramuzza, D. (2014). SVO : Fast semi-direct monocular visual odometry. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 15–22.

Fox, D., Burgard, W., Dellaert, F. et Thrun, S. (1999). Monte Carlo localization : Efficient position estimation for mobile robots. Dans *Proceedings National Conference on Artificial Intelligence and Innovative Applications of Artificial Intelligence*. p. 343–349.

Fox, D., Burgard, W. et Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, volume 4, numéro 1, p. 23–33.

Fuentes-Pacheco, J., Ruiz-Ascencio, J. et Rendón-Mancha, J. M. (2015). Visual simultaneous localization and mapping : A survey. *Artificial Intelligence Review*, volume 43, numéro 1, p. 55–81.

Gálvez-López, D. et Tardós, J. D. (2012). Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, volume 28, numéro 5, p. 1188–1197.

Garcia-Fidalgo, E. et Ortiz, A. (2015). Vision-based topological mapping and localization methods : A survey. *Robotics and Autonomous Systems*, volume 64, p. 1 – 20.

Geiger, A., Lenz, P. et Urtasun, R. (2012). Are we ready for autonomous driving ? The KITTI vision benchmark suite. Dans *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. p. 3354–3361.

Geiger, A., Ziegler, J. et Stiller, C. (2011). StereoScan : Dense 3D reconstruction in real-time. Dans *Proceedings Intelligent Vehicles Symposium*. p. 963–968.

Glover, A. J., Maddern, W. P., Milford, M. J. et Wyeth, G. F. (2010). FAB-MAP + RatSLAM : Appearance-based SLAM for multiple times of day. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 3507–3512.

Goebel, P. (2014). Winning the IROS 2014 Microsoft Kinect Challenge. https ://www.meetup.com/SV-ROS-users/pages/17825242/Winning_the_IROS2014_Microsoft_Connect_Challenge/.

Grisetti, G., Grzonka, S., Stachniss, C., Pfaff, P. et Burgard, W. (2007a). Efficient estimation of accurate maximum likelihood maps in 3D. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 3472–3478.

Grisetti, G., Kümmerle, R., Stachniss, C. et Burgard, W. (2010). A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, volume 2, numéro 4, p. 31–43.

Grisetti, G., Stachniss, C. et Burgard, W. (2007b). Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, volume 23, numéro 1, p. 34–46.

Gutierrez-Gomez, D., Mayol-Cuevas, W. et Guerrero, J. J. (2016). Dense RGB-D visual odometry using inverse depth. *Robotics and Autonomous Systems*, volume 75, p. 571–583.

Harmat, A., Trentini, M. et Sharf, I. (2015). Multi-camera tracking and mapping for unmanned aerial vehicles in unstructured environments. *Journal of Intelligent & Robotic Systems*, volume 78, numéro 2, p. 291–317.

Herrera, C. D., Kim, K., Kannala, J., Pulli, K. et Heikkilä, J. (2014). DT-SLAM : Deferred triangulation for robust SLAM. Dans *Proceedings IEEE International Conference on 3D Vision*, volume 1. p. 609–616.

Hess, W., Kohler, D., Rapp, H. et Andor, D. (2016). Real-time loop closure in 2D LIDAR SLAM. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 1271–1278.

Ho, K. L. et Newman, P. (2006). Loop closure detection in SLAM by combining visual and spatial appearance. *Robotics and Autonomous Systems*, volume 54, numéro 9, p. 740–749.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. et Burgard, W. (2013). Octomap : An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, volume 34, numéro 3, p. 189–206.

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D. et Roy, N. (2011). Visual odometry and mapping for autonomous flight using an RGB-D camera. Dans *Proceedings International Symposium on Robotics Research*. p. 235–252.

Johannsson, H., Kaess, M., Fallon, M. et Leonard, J. (2013). Temporally scalable visual SLAM using a reduced pose graph. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 54–61.

Johannsson, H., Kaess, M., Fallon, M. et Leonard, J. J. (2012). Temporally scalable visual SLAM using a reduced pose graph. Dans *RSS Workshop on Long-Term Operation of Autonomous Robotic Systems in Changing Environments*. p. 54–61.

Kähler, O., Prisacariu, V. A. et Murray, D. W. (2016). Real-time large-scale dense 3D reconstruction with loop closure. Dans *Proceedings European Conference on Computer Vision*. p. 500–516.

Kerl, C., Sturm, J. et Cremers, D. (2013). Dense visual SLAM for RGB-D cameras. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 2100–2106.

Kim, B., Kaess, M., Fletcher, L., Leonard, J., Bachrach, A., Roy, N. et Teller, S. (2010). Multiple relative pose graphs for robust cooperative mapping. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 3185–3192.

Klein, G. et Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. Dans *Proceedings IEEE and ACM International Symposium on Mixed and Augmented Reality*, IEEE. p. 225–234.

Kohlbrecher, S., Meyer, J., von Stryk, O. et Klingauf, U. (2011). A flexible and scalable SLAM system with full 3D motion estimation. Dans *Proceedings IEEE International Symposium on Safety, Security and Rescue Robotics*. p. 155–160.

Kohlbrecher, S., Rose, C., Koert, D., Manns, P., Kunz, F., Wartusch, B., Daun, K., Stumpf, A. et von Stryk, O. (2016). *RoboCup Rescue 2016 Team Description Paper Hector Darmstadt* (Rapport technique). Technische Universitaet Darmstadt.

Konolige, K. (1998). Small vision systems : Hardware and implementation. Dans Yoshiaki Shirai, S. H., *Robotics Research*. Springer, London, UK, p. 203–212.

Konolige, K. et Bowman, J. (2009). Towards lifelong visual maps. Dans *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. p. 1156–1163.

Konolige, K., Bowman, J., Chen, J., Mihelich, P., Calonder, M., Lepetit, V. et Fua, P. (2010). View-based maps. *International Journal of Robotics Research*, volume 29, numéro 8, p. 941–957.

Konolige, K., Marder-Eppstein, E. et Marthi, B. (2011). Navigation in hybrid metric-topological maps. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 3041–3047.

Krajník, T., Fentanes, J. P., Hanheide, M. et Duckett, T. (2016). Persistent localization and life-long mapping in changing environments using the frequency map enhancement. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 4558–4563.

Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K. et Burgard, W. (2011). g2o : A general framework for graph optimization. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 3607–3613.

Labbé, M. et Michaud, F. (2013). Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, volume 29, numéro 3, p. 734–745.

Labbé, M. et Michaud, F. (2014). Online global loop closure detection for large-scale multi-session graph-gased SLAM. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 2661–2666.

Labbé, M. et Michaud, F. (2017). Long-term online multi-session graph-based splam with memory management. *Autonomous Robots*, volume 42, numéro 6, p. 1133–1150.

Laniel, S., Létourneau, D., Labbé, M., Grondin, F., Polgar, J. et Michaud, F. (2017). Adding navigation, artificial audition and vital sign monitoring capabilities to a telepresence mobile robot for remote home care applications. Dans *Proceedings International Conference on Rehabilitation Robotics*. p. 809–811.

Latif, Y., Cadena, C. et Neira, J. (2013). Robust loop closing over time for pose graph SLAM. *International Journal of Robotics Research*, volume 32, numéro 14, p. 1611—1626.

Latif, Y., Lerma, C. D. C. et Neira, J. (2012). Robust loop closing over time. Dans *Robotics : Science and Systems*. p. 233–240.

Lee, G. H., Fraundorfer, F. et Pollefeys, M. (2013). Robust pose-graph loop-closures with expectation-maximization. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 556–563.

Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R. et Furgale, P. (2015). Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, volume 34, numéro 3, p. 314–334.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, volume 60, numéro 2, p. 91–110.

Lu, F. et Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, volume 4, numéro 4, p. 333–349.

Lucas, B. D. et Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. Dans *Proceedings International Joint Conference on Artificial Intelligence*. Vancouver, BC, Canada, p. 674–679.

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B. et Konolige, K. (2010). The Office Marathon : Robust navigation in an indoor office environment. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 300–307.

McDonald, J., Kaess, M., Cadena, C., Neira, J. et Leonard, J. (2012). Real-time 6-DOF multi-session visual SLAM over large scale environments. *Robotics and Autonomous Systems*, volume 61, numéro 10, p. 1144–58.

Milford, M. et Wyeth, G. (2010). Persistent navigation and mapping using a biologically inspired SLAM system. *International Journal of Robotics Research*, volume 29, numéro 9, p. 1131–53.

Moore, T. et Stouch, D. (2014). A generalized extended Kalman filter implementation for the Robot Operating System. Dans *Proceedings International Conference on Intelligent Autonomous Systems*. Springer, Shanghai, China, p. 335–348.

Muja, M. et Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. Dans *Proceedings International Conference on Computer Vision Theory and Application*. p. 331–340.

Mur-Artal, R., Montiel, J. M. M. et Tardos, J. D. (2015). ORB-SLAM : A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, volume 31, numéro 5, p. 1147–1163.

Mur-Artal, R. et Tardós, J. D. (2017). ORB-SLAM2 : An open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, volume 33, numéro 5, p. 1255–1262.

Pire, T., Fischer, T., Castro, G., De Cristóforis, P., Civera, J. et Jacobo Berlles, J. (2017). S-PTAM : Stereo Parallel Tracking and Mapping. *Robotics and Autonomous Systems*, volume 93, p. 27 – 42.

Pirker, K., Ruther, M. et Bischof, H. (2011). CD SLAM – Continuous localization and mapping in a dynamic world. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 3990–3997.

Pizzoli, M., Forster, C. et Scaramuzza, D. (2014). REMODE : Probabilistic, monocular dense reconstruction in real time. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 2609–2616.

Pomerleau, F., Colas, F., Siegwart, R. et Magnenat, S. (2013). Comparing ICP variants on real-world data sets. *Autonomous Robots*, volume 34, numéro 3, p. 133–148.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. et Ng, A. (2009). ROS : An open-source Robot Operating System. Dans *ICRA Workshop on Open Source Software*.

Rublee, E., Rabaud, V., Konolige, K. et Bradski, G. (2011). ORB : An efficient alternative to SIFT or SURF. Dans *Proceedings IEEE International Conference on Computer Vision*. p. 2564–2571.

Rusu, R. B. et Cousins, S. (2011). 3D is here : Point Cloud Library (PCL). Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 1–4.

Scaramuzza, D. et Fraundorfer, F. (2011). Visual odometry [tutorial]. *IEEE Robotics & Automation Magazine*, volume 18, numéro 4, p. 80–92.

Schlegel, D., Colosi, M. et Grisetti, G. (2017). ProSLAM : Graph SLAM from a programmer's perspective. *arXiv preprint arXiv :1709.04377*.

Schneider, T., Dymczyk, M., Fehr, M., Egger, K., Lynen, S., Gilitschenski, I. et Siegwart, R. (2018). maplab : An open framework for research in visual-inertial mapping and localization. *IEEE Robotics and Automation Letters*, volume 3, numéro 3, p. 1418–1425.

Shi, J. *et coll.* (1994). Good features to track. Dans *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. p. 593–600.

Shiffrin, R. M. (2003). Modeling memory and perception. *Cognitive Science*, p. 341–78.

Sivic, J. et Zisserman, A. (2003). Video Google : A text retrieval approach to object matching in videos. Dans *Proceedings International Conference on Computer Vision*. p. 1470–1478.

Stachniss, C. (2009). *Robotic Mapping and Exploration*, volume 55. Springer Science & Business Media.

Stachniss, C., Leonard, J. J. et Thrun, S. (2016). Simultaneous localization and mapping. Dans Bruno Siciliano, O. K., *Springer Handbook of Robotics*, chapitre 46. Springer, Cham, Switzerland, p. 1153–1176.

Steux, B. et El Hamzaoui, O. (2010). tinySLAM : A SLAM algorithm in less than 200 lines C-language program. Dans *Proceedings IEEE International Conference on Control Automation Robotics & Vision*. p. 1975–1979.

Sturm, J., Engelhard, N., Endres, F., Burgard, W. et Cremers, D. (2012). A benchmark for the evaluation of RGB-D SLAM systems. Dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. p. 573–580.

Sun, K., Mohta, K., Pfrommer, B., Watterson, M., Liu, S., Mulgaonkar, Y., Taylor, C. J. et Kumar, V. (2018). Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, volume 3, numéro 2, p. 965–972.

Sunderhauf, N. et Protzel, P. (2012). Towards a robust back-end for pose graph SLAM. Dans *Proceedings IEEE International Conference on Robotics and Automation*. p. 1254–1261.

Thrun, S. (2002). Robotic mapping : A survey. Dans *Exploring Artificial Intelligence in the New Millennium*, volume 1. Morgan Kaufmann Publishers, p. 1–35.

Thrun, S., Burgard, W. et Fox, D. (2005). *Probabilistic Robotics*. The MIT Press, 667 p.

Thrun, S. et Montemerlo, M. (2006). The graph SLAM algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, volume 25, numéro 5-6, p. 403–429.

Valencia, R., Morta, M., Andrade-Cetto, J. et Porta, J. M. (2013). Planning reliable paths with Pose SLAM. *IEEE Transactions on Robotics*, volume 29, numéro 4, p. 1050–1059.

Vincent, R., Limketkai, B. et Eriksen, M. (2010). Comparison of indoor robot localization techniques in the absence of GPS. Dans *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV*, International Society for Optics and Photonics. Volume 7664. p. 1Z.

Walcott-Bryant, A., Kaess, M., Johannsson, H. et Leonard, J. J. (2012). Dynamic pose graph SLAM : Long-term mapping in low dynamic environments. Dans *Proceedings IEEE/RSJ Conference on Intelligent Robots and Systems*. p. 1871–1878.

Whelan, T., Kaess, M., Johannsson, H., Fallon, M., Leonard, J. J. et McDonald, J. (2015). Real-time large-scale dense RGB-D SLAM with volumetric fusion. *The International Journal of Robotics Research*, volume 34, numéro 4-5, p. 598–626.

Whelan, T., Salas-Moreno, R. F., Glocker, B., Davison, A. J. et Leutenegger, S. (2016). ElasticFusion : Real-time dense SLAM and light source estimation. *The International Journal of Robotics Research*, volume 35, numéro 14, p. 1697–1716.

Xu, W. et Mulligan, J. (2010). Performance evaluation of color correction approaches for automatic multi-view image and video stitching. Dans *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. p. 263–270.

Yi, L., Fei, G., Tong, Q., Wenliang, G., Tianbo, L., William, W., Zhenfei, Y. et Shaojie, S. (2017). Autonomous aerial navigation using monocular visual-inertial fusion. *Journal of Field Robotics*, volume 35, numéro 1, p. 23–51.

Zhang, J. et Singh, S. (2017). Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, volume 41, numéro 2, p. 401–416.

Zollhöfer, M., Stotko, P., Görlitz, A., Theobalt, C., Nießner, M., Klein, R. et Kolb, A. (2018). State of the art on 3D reconstruction with RGB-D cameras. Dans *Computer Graphics Forum*, Wiley Online Library. Volume 37. p. 625–652.